Progetto di Machine Learning

Antonio Grillo, Giacomo Motta, Gabriele Pandini

25 Gennaio 2023

Dataset Diabetes

Questo set di dati proviene originariamente dal National Institute of Diabetes and Digestive and Kidney Diseases. L'obiettivo è di prevedere diagnosticamente se un paziente ha il diabete, sulla base di determinate misurazioni diagnostiche. I dati sono tratti da un database più ampio. In particolare, tutti gli individui del dataset sono femmine di almeno 21 anni di origine Pima (un gruppo di nativi americani viventi in un territorio che attualmente comprende l'Arizona centrale e meridionale).

Dal dataset nel file (`diabetes.csv`) possiamo trovare diverse variabili, alcune delle quali sono indipendenti (diverse variabili predittive mediche) e solo una variabile è dipendente dall'obiettivo (risultato).

Il dataset è reperibile al seguente link: https://www.kaggle.com/datasets/whenamancodes/predict-diabities.

Analisi degli attributi presenti nel dataset:

- `Pregnancies`: esprime il numero di gravidanze.
- `Glucose`: esprime il livello di glucosio nel sangue. Soggetti sani hanno valori di glucosio nel sangue compresi tra 70 mg/dl e 140 mg/dl circa (1).
- `BloodPressure`: esprime il livello della pressione del sangue. Soggetti sani hanno valori di pressione sanguigna tra 80 mmHg e 129 mmHg circa (2).
- `SkinThickness`: esprime lo spessore della pelle.
- `Insulin`: esprime il livello di insulina nel sangue. In Soggetti sani varia tra 5 e 125 micr.UI/ml (3).
- `BMI`: esprime l'indice di massa corporea, in soggetti sani è compreso nell'intervallo 18,5-25 (4).
- `DiabetesPedigreeFunction`: esprime la probabilità di avere il diabete in base alla storia familiare (5).
- `Age`: esprime l'eta del soggetto.
- `Outcome`: indica se il soggetto è diabetico oppure no.

Librerie ed import utilizzati

Segue l'elenco delle librerie da installare per lo scopo del progetto:

```
install.packages(c("FactoMineR", "factoextra"))
install.packages("lattice")
install.packages("dplyr")
install.packages("lattice")
install.packages("rpart")
install.packages("rattle")
install.packages("rpart.plot")
install.packages("RColorBrewer")
install.packages("pROC")
```

```
install.packages("caret")
install.packages("ROCR")
```

Segue invece l'elenco delle librerie da importare:

```
library(dplyr)
```

```
##
## Caricamento pacchetto: 'dplyr'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     filter, lag
```

```
## I seguenti oggetti sono mascherati da 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(FactoMineR)
library(factoextra)
```

```
## Caricamento del pacchetto richiesto: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(lattice)
library(rpart)
library(rattle)
```

```
## Caricamento del pacchetto richiesto: tibble
```

```
## Caricamento del pacchetto richiesto: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(rpart.plot)
library(RColorBrewer)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Caricamento pacchetto: 'pROC'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     cov, smooth, var
```

```
library(caret)
library(ROCR)
```

Lettura e preprocessing del dataset

Viene letto il dataset tramite le seguenti istruzioni. Viene utilizzata la funzione `str` per avere un idea iniziale del dataset.

```
dataset = read.csv("diabetes.csv", header = TRUE)
str(dataset)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

Dal risultato della funzione `str` si può notare che sono presenti diversi attributi che hanno valore `0`. Di norma, per i valori `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin` e `BMI`, non può accadere, quindi ciò determina la mancaza del valore dell'attributo. Di conseguenza viene controllata l'effettiva numerosità di istanze, per ogni attributo, che possiedono valori a `0`. Il risultato viene salvato nel dataframe `df` e in seguito mostrato.

```
df = data.frame(colSums(dataset==0))
df
```

```
##                           colSums.dataset....0.
## Pregnancies                                 111
## Glucose                                       5
## BloodPressure                                35
## SkinThickness                               227
## Insulin                                     374
## BMI                                          11
## DiabetesPedigreeFunction                      0
## Age                                           0
## Outcome                                     500
```

Analizzando i risultati ottenuti è stato deciso di togliere le istanze che possiedono valori di `Glucose` e/o `BMI` uguale a `0` dal momento che risulterebbero circa il 2% del dataset. Successivamente verrano trattate anche le istanze che possiedono valori nulli per attributi `BloodPressure`, `SkinThickness` e `Insulin`.

La seguente funzione rimuove le istanze con valori di `Glucose` e/o `BMI` uguale a `0`.

```
dataset = filter(dataset, Glucose > 0 & BMI > 0)
```

In seguito vengono analizzati nuovamente gli attributi con valori pari a `0`.

```
df = data.frame(colSums(dataset==0))
df
```

```
##                           colSums.dataset....0.
## Pregnancies                                 108
## Glucose                                       0
## BloodPressure                                28
## SkinThickness                               218
## Insulin                                     360
## BMI                                           0
## DiabetesPedigreeFunction                      0
## Age                                           0
## Outcome                                     488
```

Come si può notare dai risultati ottenuti, le voci `BloodPressure`, `SkinThickness` e `Insulin` possiedono numerose istanze con valori nulli. Rimuoverle potrebbe influire seriamente sull'utilizzabilità del dataset. Dunque, si è preferito sostituire tali valori nulli con la media dei relativi attributi (si noti che la media viene calcolata facendo uso delle istanze che non hanno valori nulli).

```
df = data.frame(colSums(dataset==0))
meanGlucose = (mean(dataset$Glucose) * nrow(dataset)) /
  (nrow(dataset) - df['Glucose',])
meanBloodPressure = (mean(dataset$BloodPressure) *
                       nrow(dataset)) / (nrow(dataset) -
                                           df['BloodPressure',])
meanSkinThickness = (mean(dataset$SkinThickness) * nrow(dataset))/
  (nrow(dataset) - df['SkinThickness',])
meanInsulin = (mean(dataset$Insulin) * nrow(dataset)) /
              (nrow(dataset)-df['Insulin',])
meanBMI = (mean(dataset$BMI) * nrow(dataset)) /
          (nrow(dataset)-df['BMI',])

dataset$Glucose[dataset$Glucose == 0] = meanGlucose
dataset$BloodPressure[dataset$BloodPressure == 0] =
  meanBloodPressure
dataset$SkinThickness[dataset$SkinThickness == 0] =
  as.integer(meanSkinThickness)
dataset$Insulin[dataset$Insulin == 0] = meanInsulin
dataset$BMI[dataset$BMI == 0] = meanBMI
summary(dataset)
```

```
##   Pregnancies        Glucose       BloodPressure   SkinThickness
##  Min.   : 0.000   Min.   : 44.00   Min.   : 24.0   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.75   1st Qu.: 64.0   1st Qu.:25.00
##  Median : 3.000   Median :117.00   Median : 72.0   Median :29.00
##  Mean   : 3.851   Mean   :121.94   Mean   : 72.4   Mean   :29.12
##  3rd Qu.: 6.000   3rd Qu.:141.00   3rd Qu.: 80.0   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.00   Max.   :122.0   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   : 14.0   Min.   :18.20   Min.   :0.0780           Min.   :21.00
##  1st Qu.:120.0   1st Qu.:27.50   1st Qu.:0.2440           1st Qu.:24.00
##  Median :156.1   Median :32.30   Median :0.3770           Median :29.00
##  Mean   :156.1   Mean   :32.45   Mean   :0.4731           Mean   :33.31
##  3rd Qu.:156.1   3rd Qu.:36.60   3rd Qu.:0.6275           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##     Outcome
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3511
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```
df = data.frame(colSums(dataset == 0))
df
```

```
##                           colSums.dataset....0.
## Pregnancies                                 108
## Glucose                                       0
## BloodPressure                                 0
## SkinThickness                                 0
## Insulin                                       0
## BMI                                           0
## DiabetesPedigreeFunction                      0
```

```
## Age                                         0
## Outcome                                   488
```

In seguito, per comodità, vengono convertiti i valori assunti della variabile target `Outcome` in `Yes` per il valore `1` e `No` per il valore `0`.

```
dataset$Outcome = ifelse(dataset$Outcome == "1", "Yes", "No")
```

Infine, converto la variabile `Outcome` nel tipo `factor`:

```
dataset$Outcome = factor(dataset$Outcome)
str(dataset)
```

```
## 'data.frame':    752 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 4 ...
##  $ Glucose                 : num  148 85 183 89 137 116 78 115 197 110 ...
##  $ BloodPressure           : num  72 66 64 66 40 ...
##  $ SkinThickness           : int  35 29 29 23 35 29 32 29 45 29 ...
##  $ Insulin                 : num  156 156 156 94 168 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 37.6 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 30 ...
##  $ Outcome                 : Factor w/ 2 levels "No","Yes": 2 1 2 1 2 1 2 1 2 1 ...
```

Creazione del training set e test set

Per la divisione del dataset originario in training set e test set, viene definita la seguente funzione:

```
split.data = function(data, p = 0.7, s = 1){
  set.seed(s)
  index = sample(1:dim(data)[1])
  train = data[index[1:floor(dim(data)[1] * p)], ]
  test = data[index[((ceiling(dim(data)[1] * p)) + 1):
                    dim(data)[1]], ]
  return(list(train=train, test=test))
}
```

Quindi viene eseguita la divisione del dataset.

```
allset = split.data(dataset, p = 0.7, s = 1)
trainset = allset$train
testset = allset$test
```

Si noti che vengono mantenuti i parametri di default della funzione. In particolare, viene assegnato il 70% del dataset al training set, mentre il restante al test set. Inoltre viene passato `s = 1` come seme per rendere riproducibili gli stessi valori casuali nel codice.

Descrizione del training set

In questa sezione verrà effettuata una analisi esplorativa del training set.

In primis, vengono mostrate le statistiche del primo ordine:

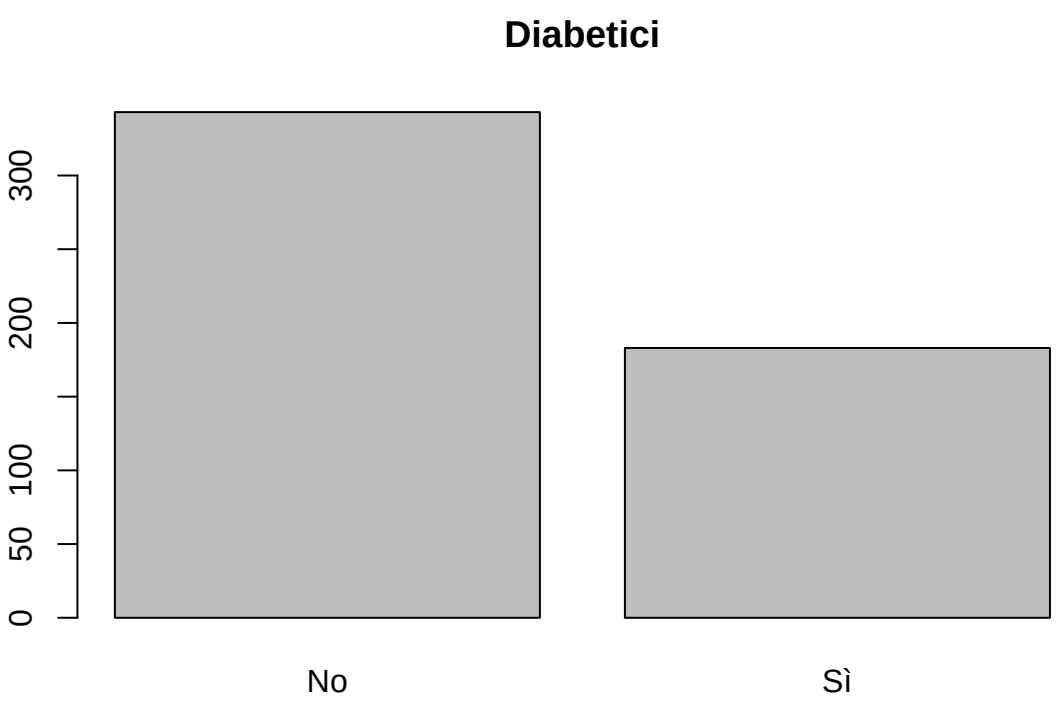
```
summary(trainset)
```

```
##   Pregnancies        Glucose      BloodPressure SkinThickness      Insulin
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24   Min.   : 7.00   Min.   : 14.0
##  1st Qu.: 1.000   1st Qu.:100.0   1st Qu.: 64   1st Qu.:25.00   1st Qu.:125.0
##  Median : 3.000   Median :119.0   Median : 72   Median :29.00   Median :156.1
```

```
##  Mean   : 3.776   Mean   :122.0   Mean   : 72   Mean   :29.25   Mean   :157.4
##  3rd Qu.: 6.000   3rd Qu.:141.8   3rd Qu.: 80   3rd Qu.:32.75   3rd Qu.:156.1
##  Max.   :15.000   Max.   :199.0   Max.   :114   Max.   :99.00   Max.   :744.0
##       BMI        DiabetesPedigreeFunction      Age        Outcome
##  Min.   :18.20   Min.   :0.0780           Min.   :21.00   No :343
##  1st Qu.:27.52   1st Qu.:0.2450           1st Qu.:24.00   Yes:183
##  Median :32.35   Median :0.3725           Median :29.00
##  Mean   :32.49   Mean   :0.4714           Mean   :33.27
##  3rd Qu.:36.67   3rd Qu.:0.6180           3rd Qu.:40.00
##  Max.   :59.40   Max.   :2.4200           Max.   :81.00
```

Segue un plot che mostra la distribuzione dei positivi e negativi, in fattori di quantità all'interno del dataset.

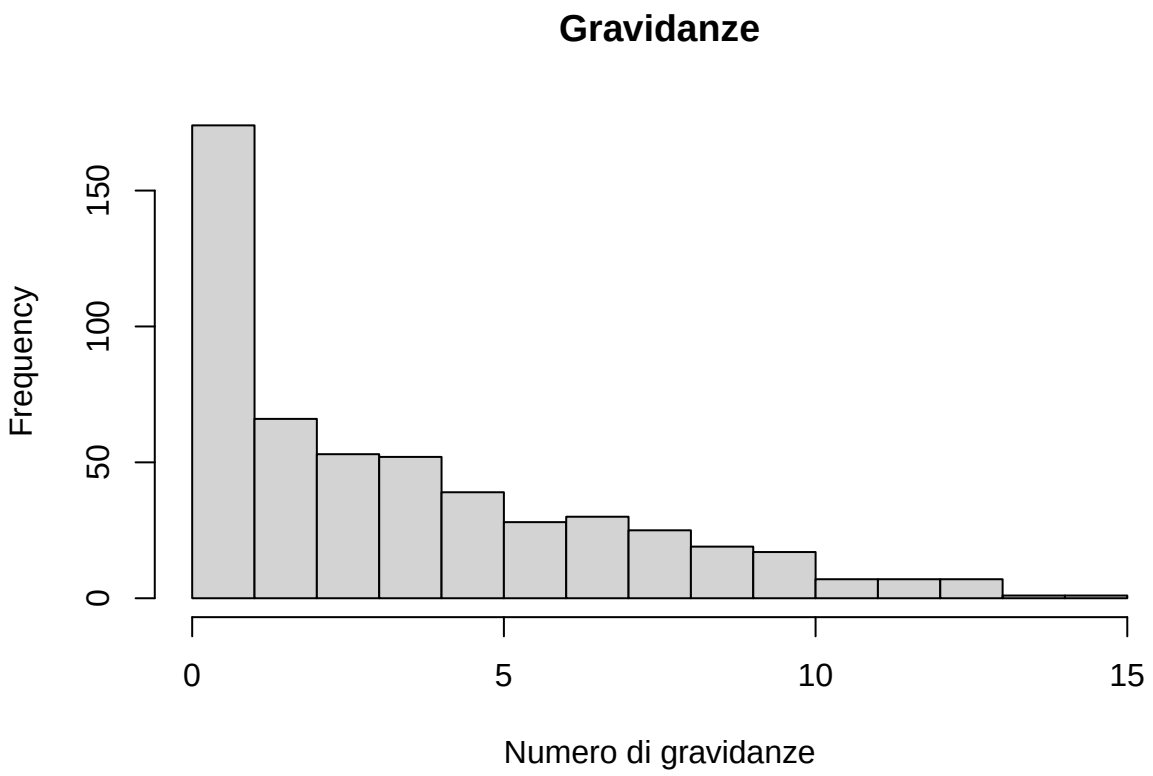
```
barplot(table(trainset$Outcome),
        main = "Diabetici",
        names = c("No", "Sì"))
```



Come si può vedere dal precedente grafico, il numero degli individui che è negativo è in numero maggiore rispetto ai positivi.

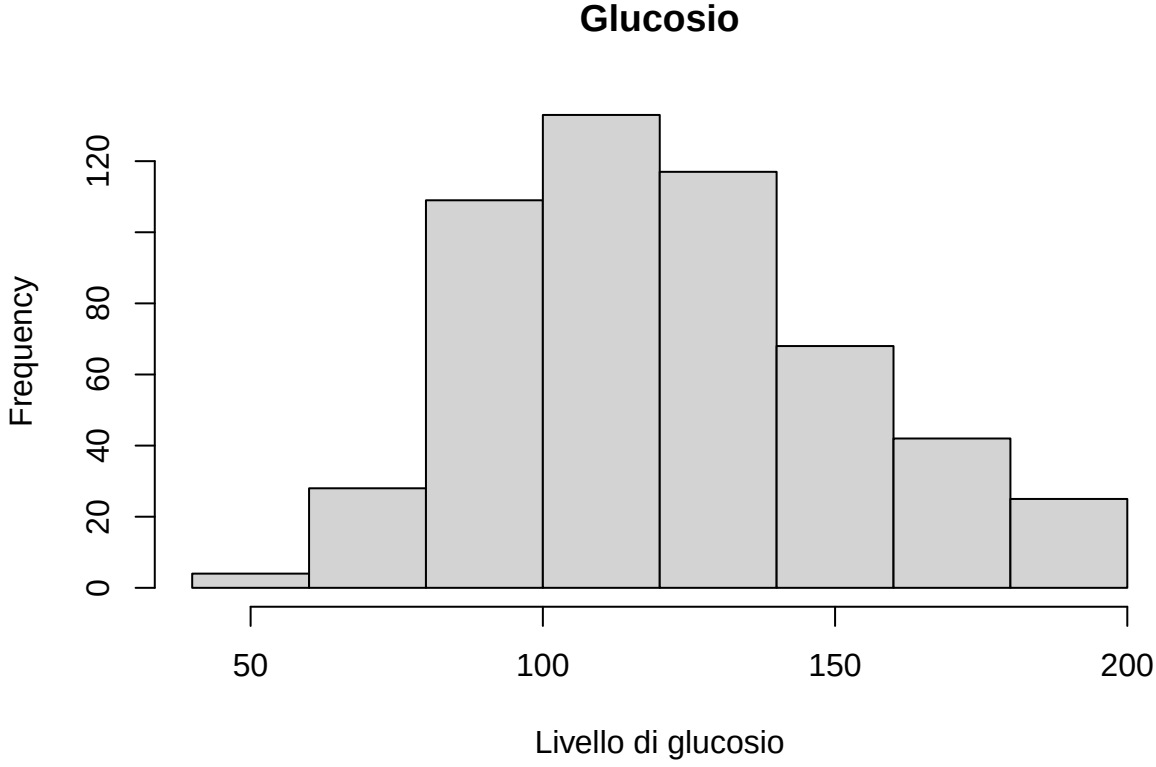
Segue il grafico che mostra l'istogramma che mostra il numeo di gravidanze degli individui:

```
hist(trainset$Pregnancies,
     main = "Gravidanze",
     xlab="Numero di gravidanze")
```



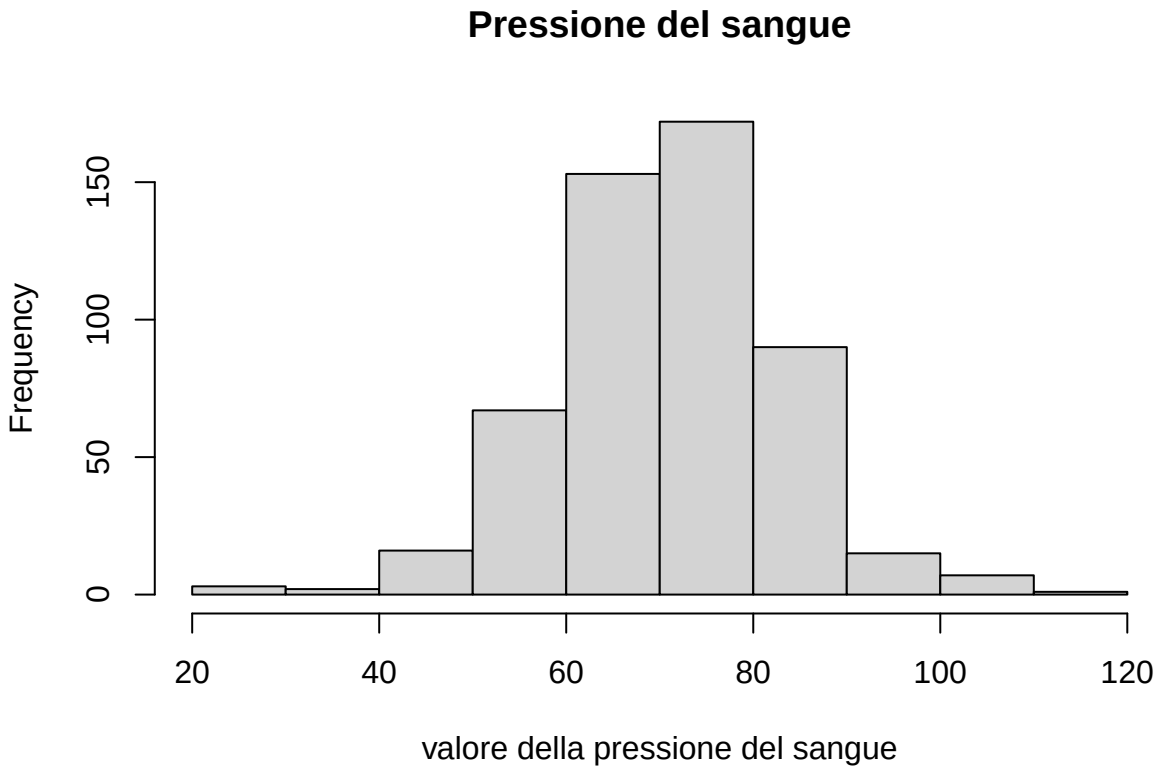
Il seguente istogramma mostra il livello di glucosio negli individui:

```
hist(trainset$Glucose,
     main = "Glucosio",
     xlab="Livello di glucosio")
```

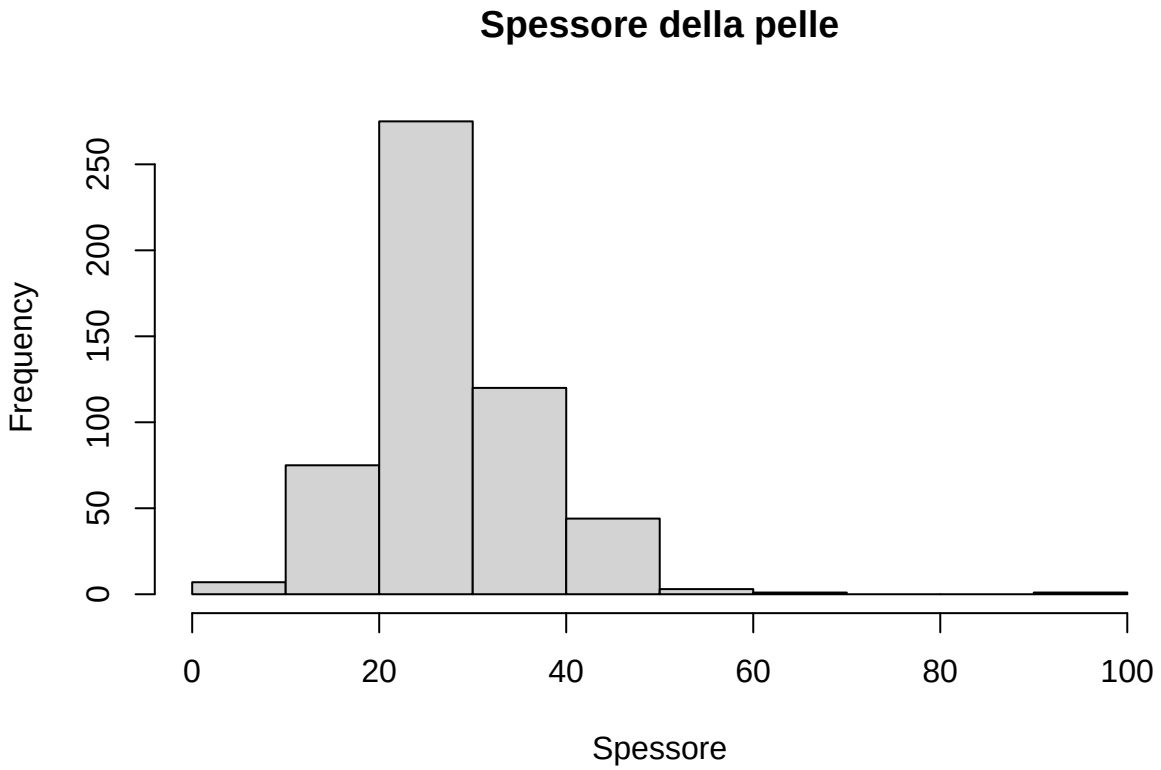


Seguono i grafici che mostrano il valore della pressioine del sangue e dello spessore della pelle:

```
hist(trainset$BloodPressure,
     main = "Pressione del sangue",
     xlab="valore della pressione del sangue")
```

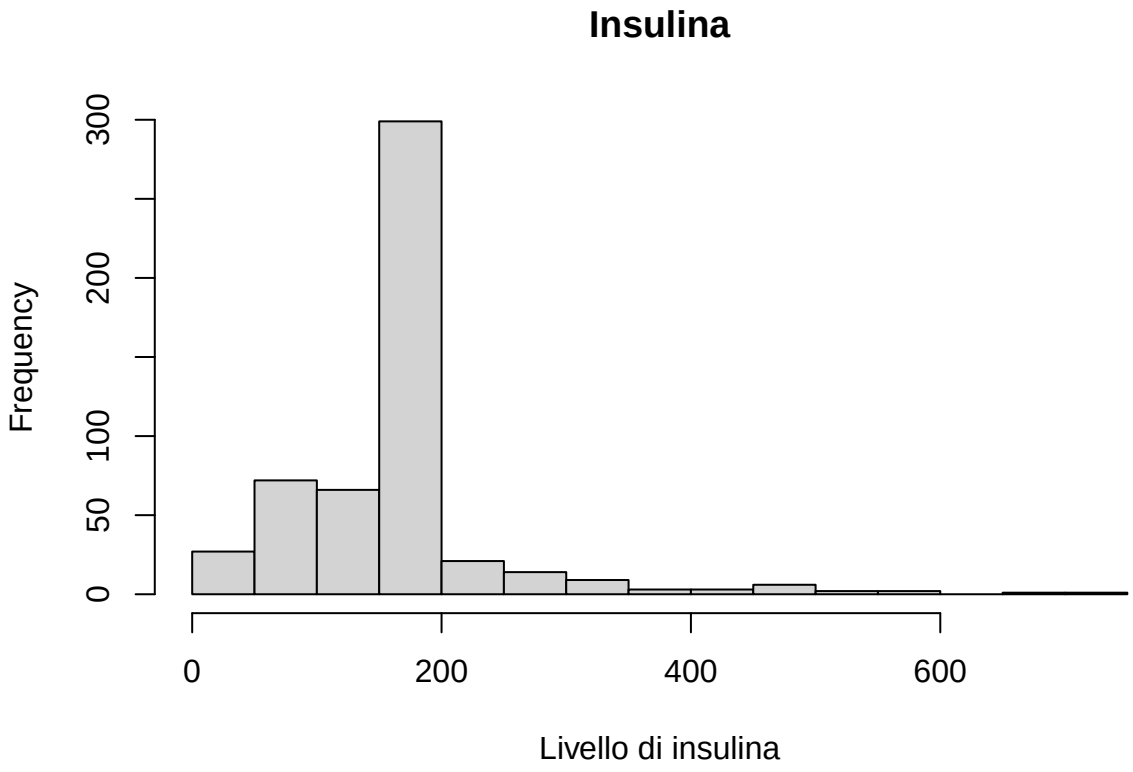



```
hist(trainset$SkinThickness,
     main = "Spessore della pelle",
     xlab="Spessore")
```

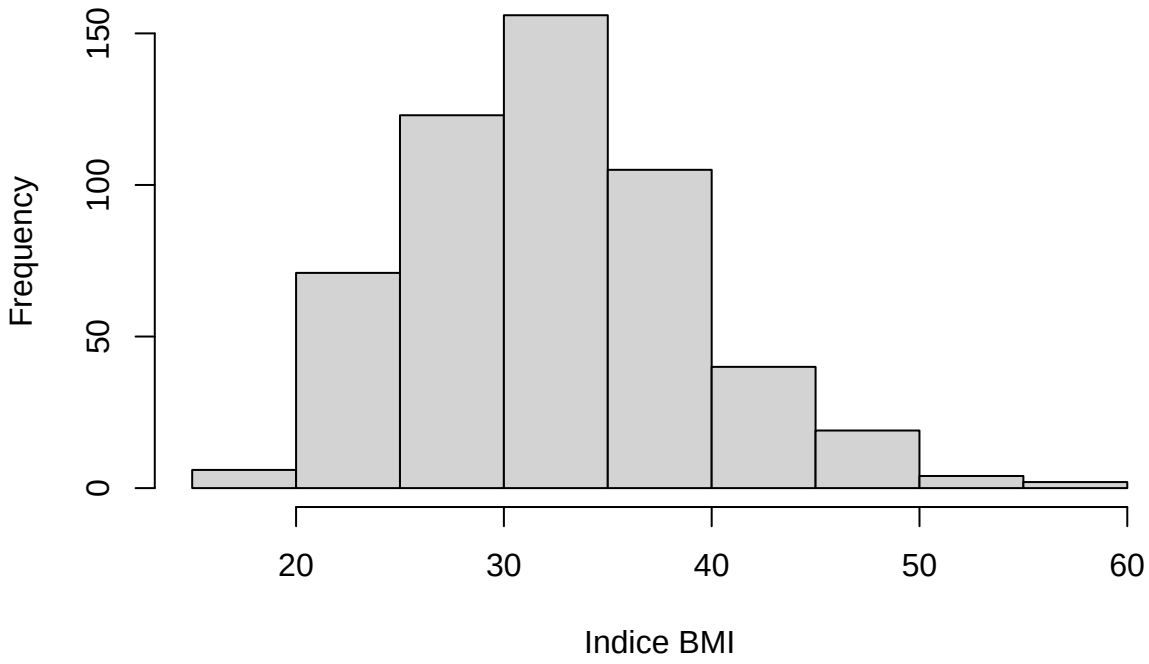


Successivamente, viene presentato l'istogramma relativo ai livelli di insulina e quello sull'indice BMI degli individui.

```
hist(trainset$Insulin,
     main = "Insulina",
     xlab="Livello di insulina")
```

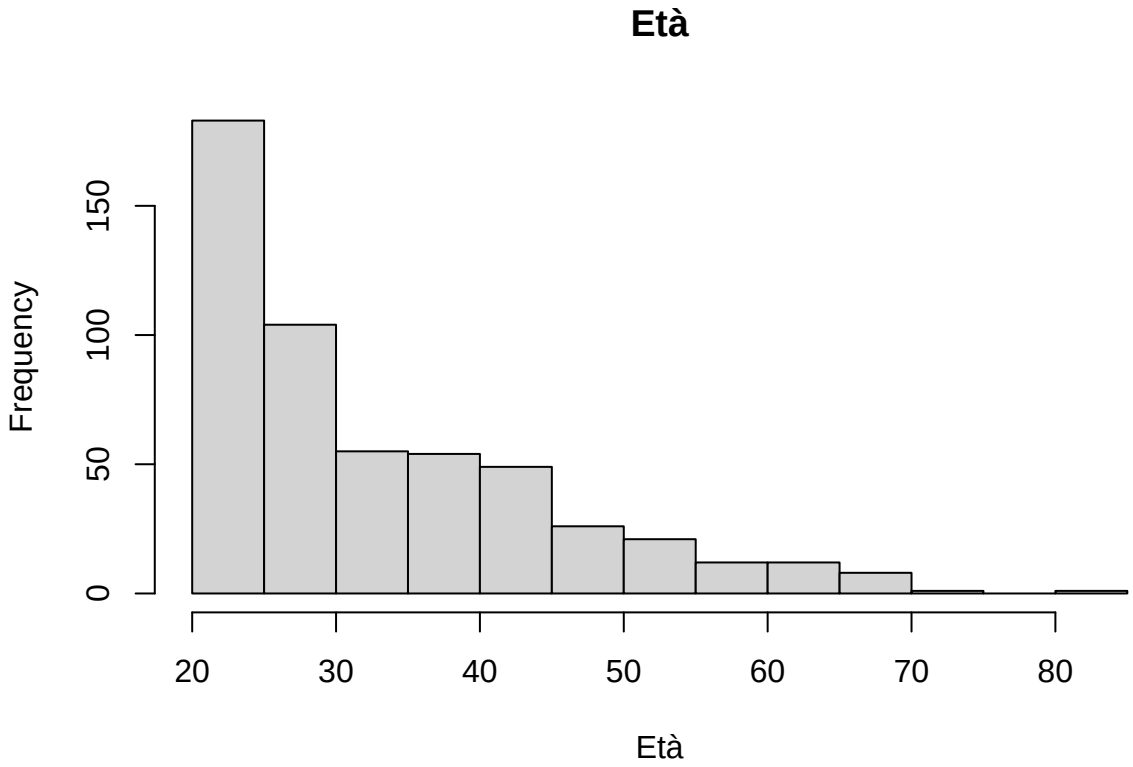



```
hist(trainset$BMI,
     main = "BMI",
       xlab="Indice BMI")
```



Inoltre, la distribuzione delle età all'interno del dataset è la seguente:

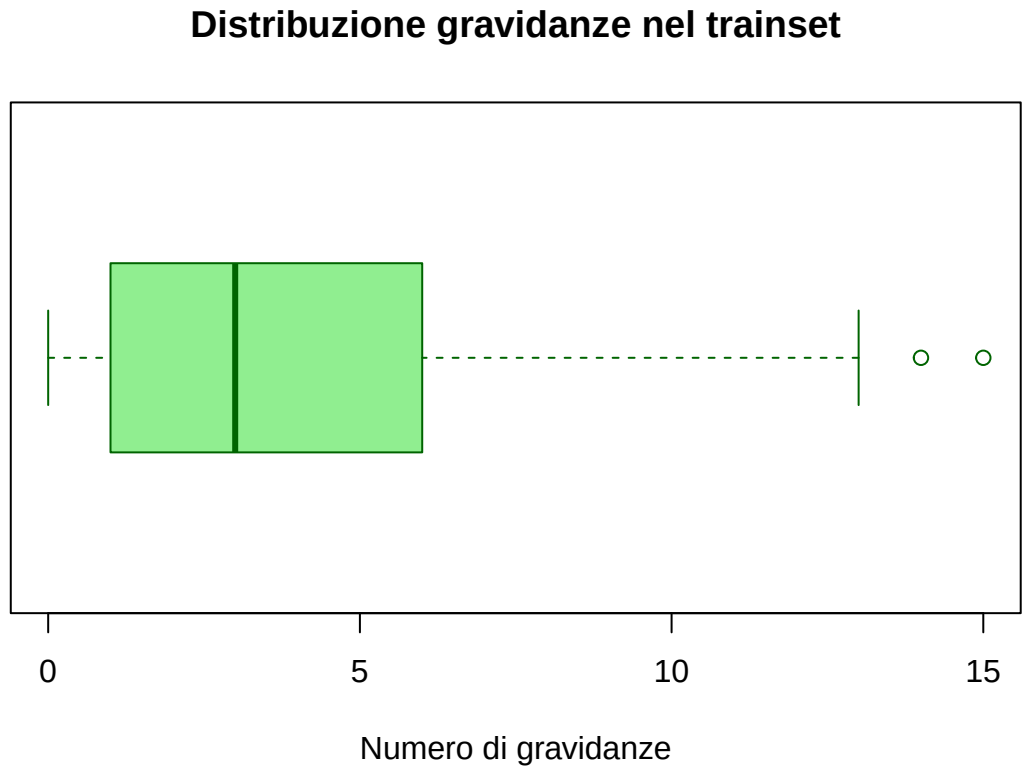
```
hist(trainset$Age,
     main = "Età",
     xlab="Età")
```



Da ciò si deduce che sono prevalenti gli individui tra i 20 e i 30 anni.

Segue il boxplot che analizza le gravidanze:

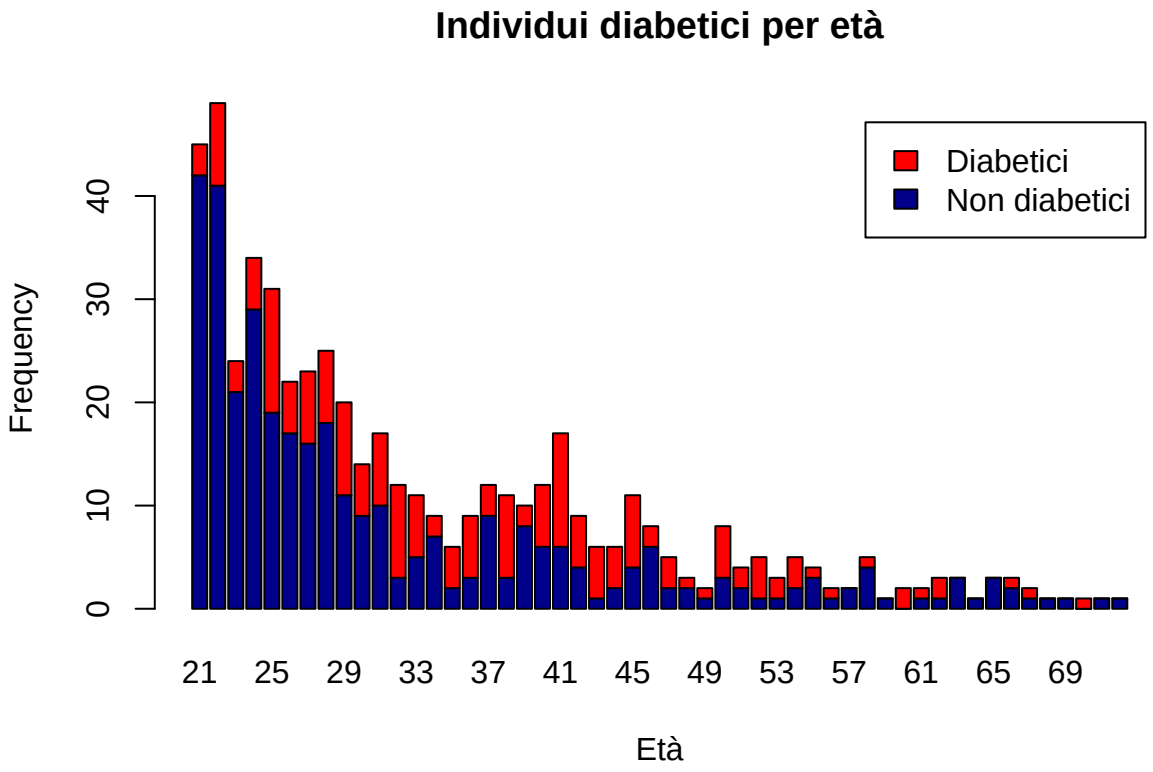
```
boxplot(trainset$Pregnancies,
        col = "lightgreen",
        border = "darkgreen",
        horizontal = TRUE,
        main = "Distribuzione gravidanze nel trainset",
        xlab = "Numero di gravidanze"
)
```



Il risultato di questo grafico è che la mediana delle nascite si trova attorno al valore 3.

In seguito, vengono confrontati i dati relativi alle età dei campioni e, per ognuna, la proporzione di positività o no al diabete.

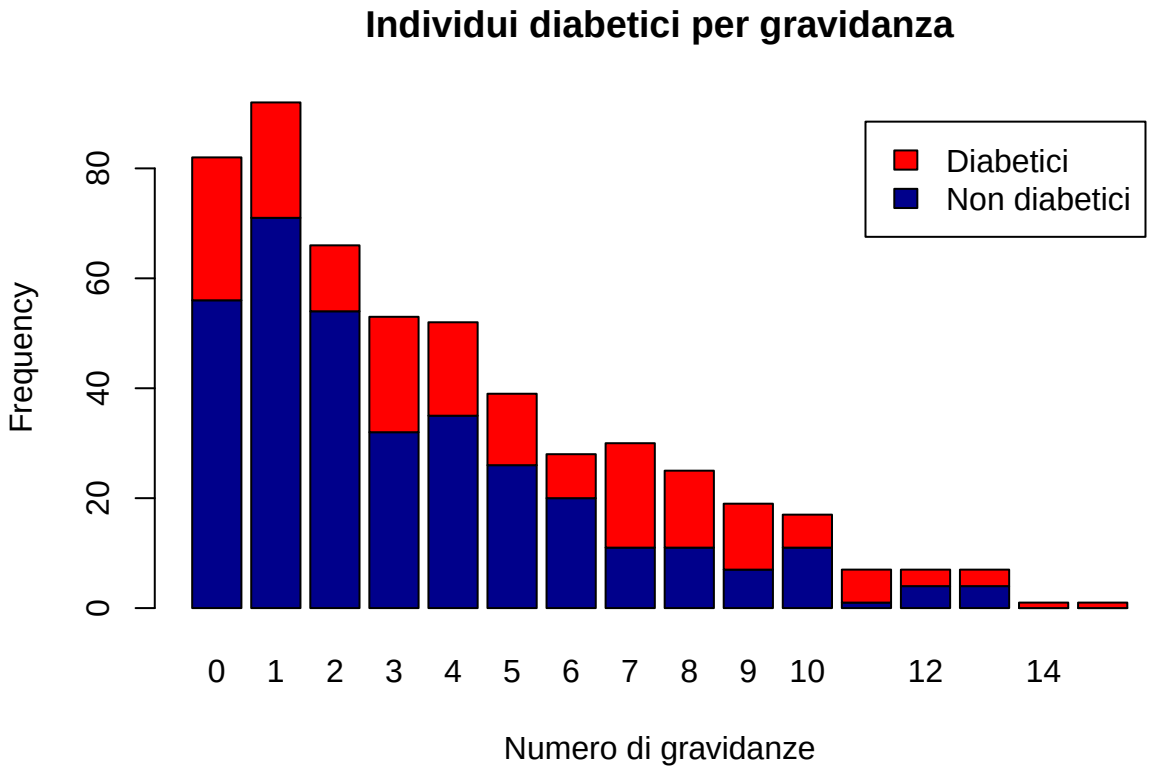
```
barplot(table(trainset$Outcome, trainset$Age),
        col=c("darkblue","red"),
        legend = c("Non diabetici", "Diabetici"),
        main = "Individui diabetici per età",
        ylab = "Frequency",
        xlab = "Età")
```



Osservando il precedente istogramma, si può notare che più aumenta l'età e più è probabile che il diabete sia presente nella popolazione.

Poi, viene confrontato il numero di gravidanze con il risultato finale.

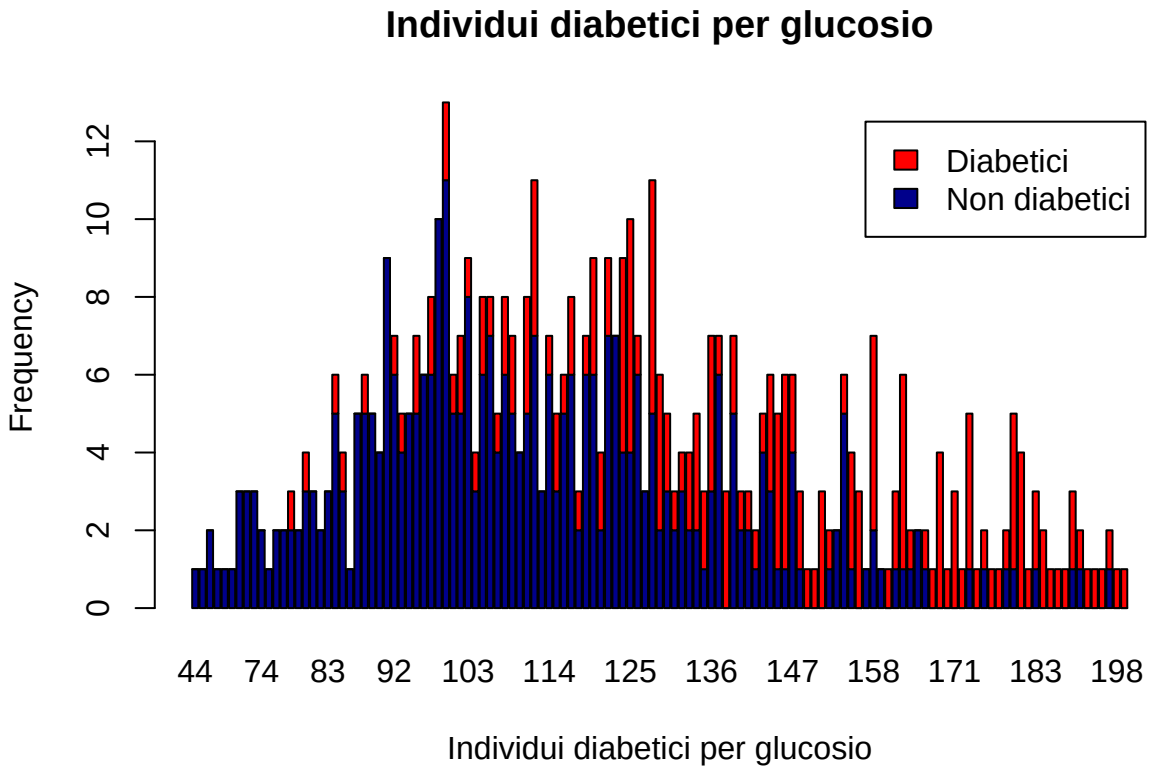
```
barplot(table(trainset$Outcome, trainset$Pregnancies),
        col=c("darkblue","red"),
        legend = c("Non diabetici", "Diabetici"),
        main = "Individui diabetici per gravidanza",
        ylab = "Frequency",
        xlab = "Numero di gravidanze")
```



Dal precedente istogramma non sembra essere incidente l'aver avuto poche gravidanze (ad esempio fino a 4) con l'essere diabetici. Tuttavia, si nota un graduale aumento delle positività con le gravidanze che sono maggiori di 5 (tranne per le eccezioni 6 e 10).

Segue il grafico che confronta i soggeti diabetici e non sulla base del livello di glucosio:

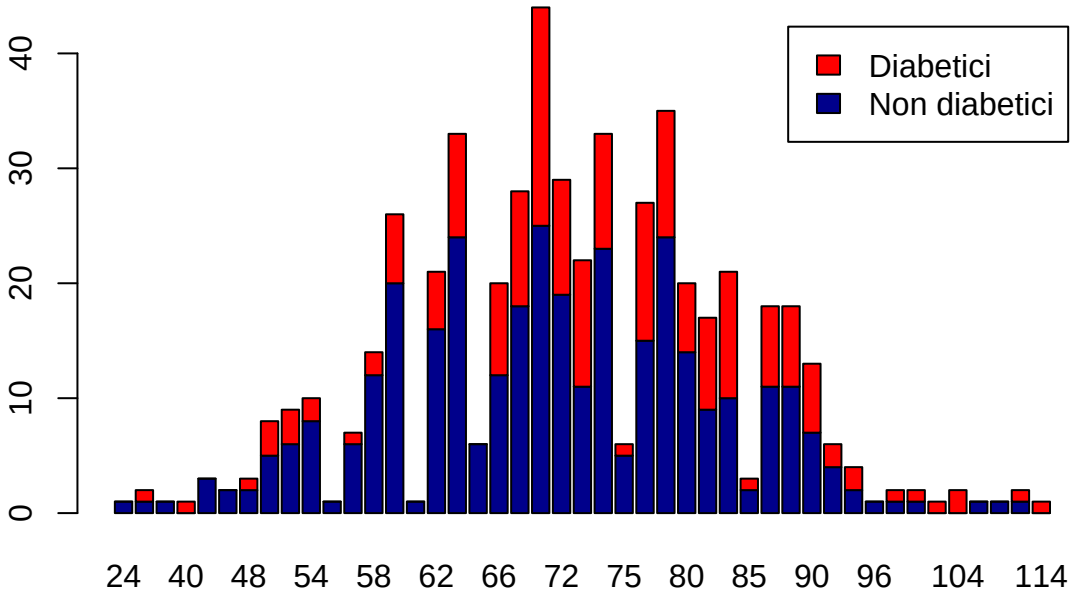
```
barplot(table(trainset$Outcome, trainset$Glucose),
        col=c("darkblue","red"),
        legend = c("Non diabetici", "Diabetici"),
        main = "Individui diabetici per glucosio",
        ylab = "Frequency",
        xlab = "Individui diabetici per glucosio")
```



Da questo grafico è deducibile che vi è prevalenza di diabetici in individui che hanno un alto livello di glucosio.

Il seguente grafico mostra la distribuzione dei paizenti diabetici e non diabetici sulla base della pressione del sangue:

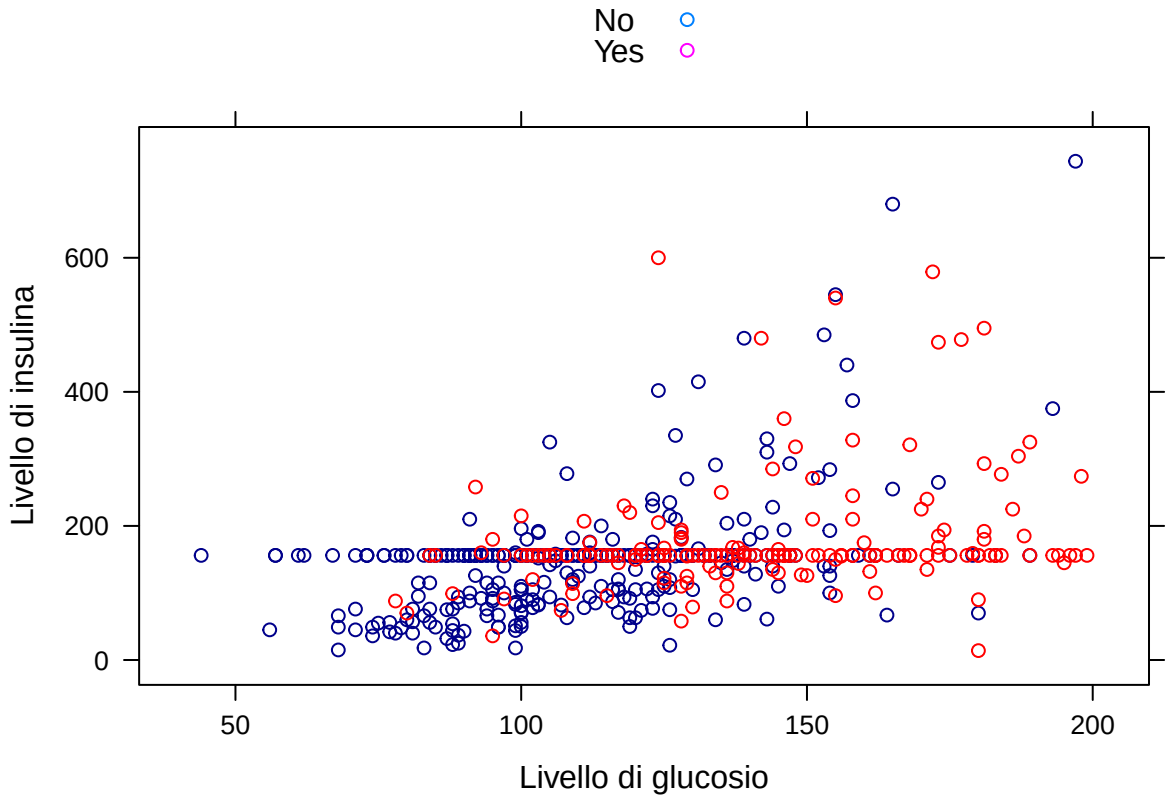
```
counts = table(trainset$Outcome, trainset$BloodPressure)
barplot(counts,
        col=c("darkblue","red"),
        legend = c("Non diabetici", "Diabetici"),
        main = "Pazienti con diabete per pressione sanguigna")
```



Dal risultato si può affermare che non è presente una correlazione tra il diabete e uno specifico range della pressione del sangue, dal momento che la proporzione di diabetici e non della maggior parte delle colonne è costante.

Confronto l'insulina con il livello di glucosio nel sangue:

```
xyplot(trainset$Insulin ~ trainset$Glucose,
       data = trainset,
       group = Outcome,
       col=c("darkblue","red"),
       auto.key = TRUE,
       ylab = "Livello di insulina",
       xlab = "Livello di glucosio")
```



Dal precedente grafo si può notare che i dati hanno un ordinamento sparso tra di loro senza una vera e propria distinzione in base alle due variabili scelte.

Analisi delle Componenti Principali

Vengono selezionate le varibili quantitative per eseguire le PCA:

```
trainset.active = trainset[, -9]
```

Quindi non viene presa in considerazione la variabile di target, ovvero `Outcome`.

Successivamente viene eseguita la PCA ed analizzati gli autovalori:

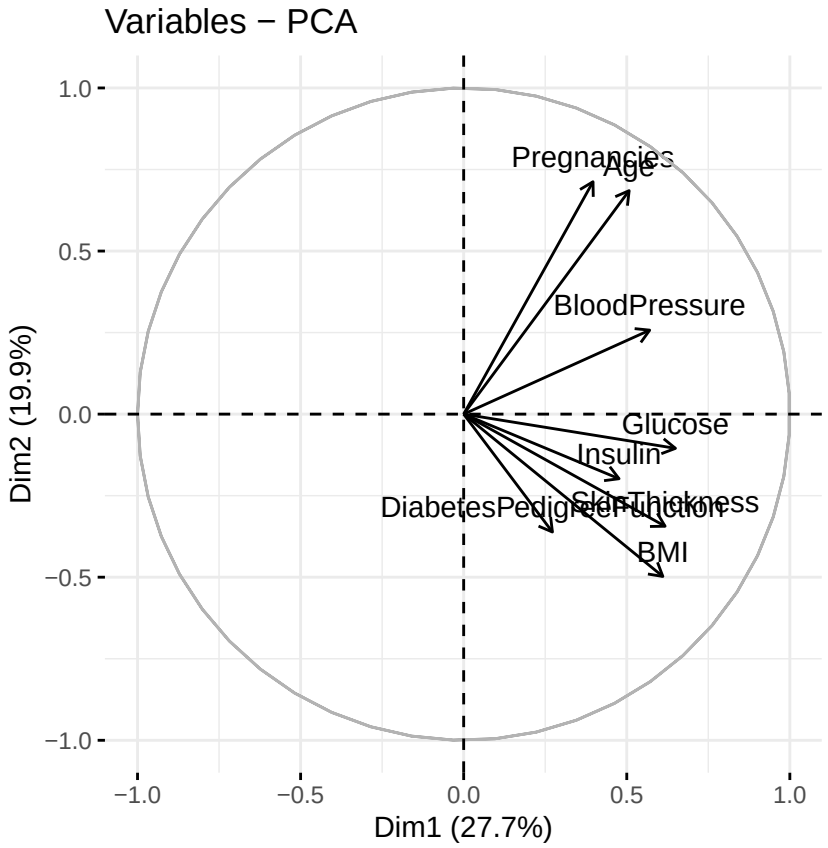
```
res.pca = PCA(trainset.active, graph = FALSE)
eig.val = get_eigenvalue(res.pca)
eig.val
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.2166858        27.708573                    27.70857
## Dim.2  1.5900557        19.875696                    47.58427
## Dim.3  1.0794809        13.493511                    61.07778
## Dim.4  0.9076429        11.345537                    72.42332
## Dim.5  0.7697737         9.622171                    82.04549
## Dim.6  0.5743495         7.179369                    89.22486
## Dim.7  0.4884353         6.105442                    95.33030
## Dim.8  0.3735761         4.669702                   100.00000
```

Per l'analisi delle componenti principali, vengono scelte le prime quattro dimensioni che coprono circa il 72% della varianza, oltre che avere un valore degli autovalori che è maggiore di uno o approssimativamente vicino.

Componenti Principali e variabili Per vedere gli effetti delle PCA sulle variabili del trainset viene effettuata una proiezione su un grafo:

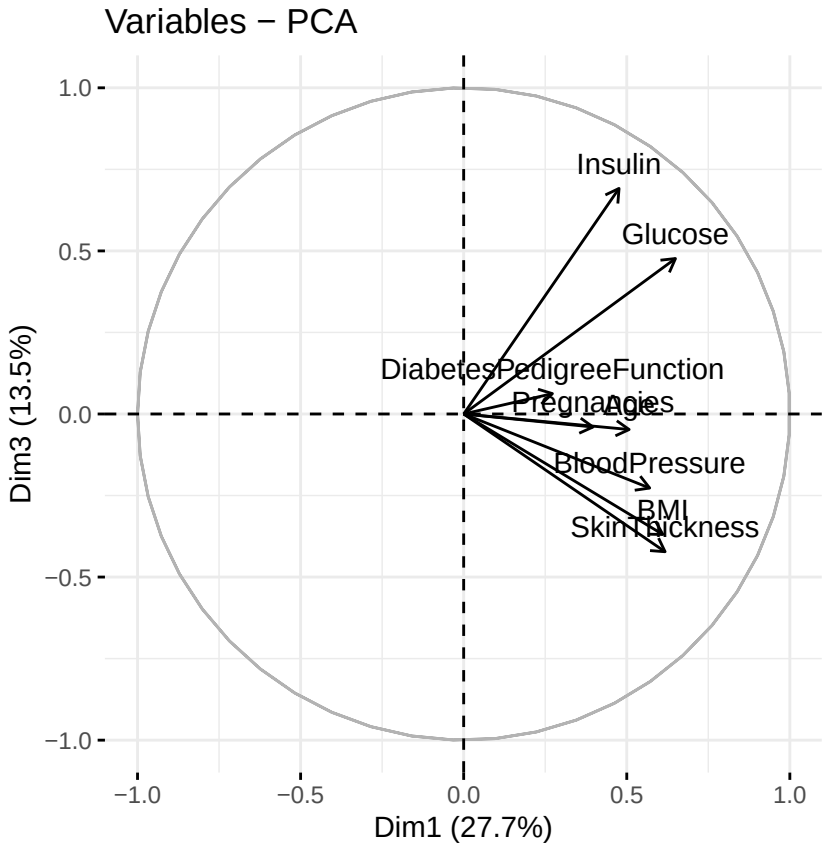
```
var = get_pca_var(res.pca)
fviz_pca_var(res.pca, col.var = "black")
```



Come si può osservare dal precedente grafico, in base alle prime due dimensioni, la maggioranza delle variabili sono positivamente correlate, visto che sono raggruppate assieme. Invece, la variabile meglio rappresentata è `Age` (che dista circa 0.86 dall'origine) perché è la più lontana dall'origine.

Analogamente per le dimensioni 1 e 3:

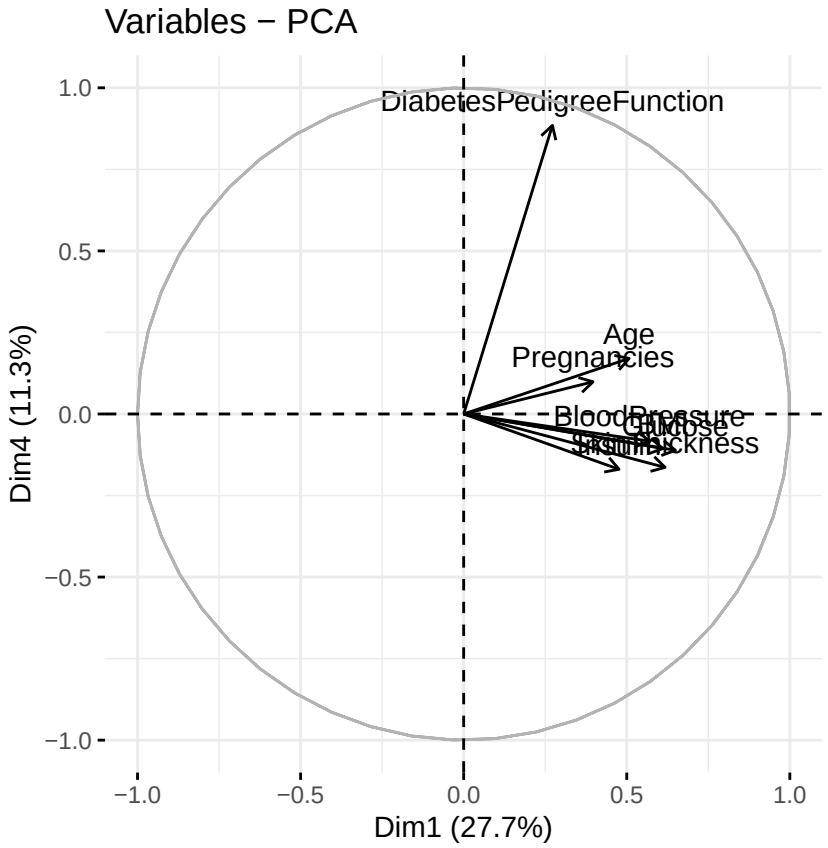
```
fviz_pca_var(res.pca, col.var = "black", axes = c(1,3))
```



Anche in questo caso si ha un risultato analogo al precedente, tuttavia la varianza coperta da queste due dimensioni è minore rispetto alle precedenti. Quindi, risulta esser meglio rappresentata la variabile `Insulin` rispetto alla componente principale 1 e 3.

Segue il grafico per le dimensioni 1 e 4:

```
fviz_pca_var(res.pca, col.var = "black", axes = c(1,4))
```

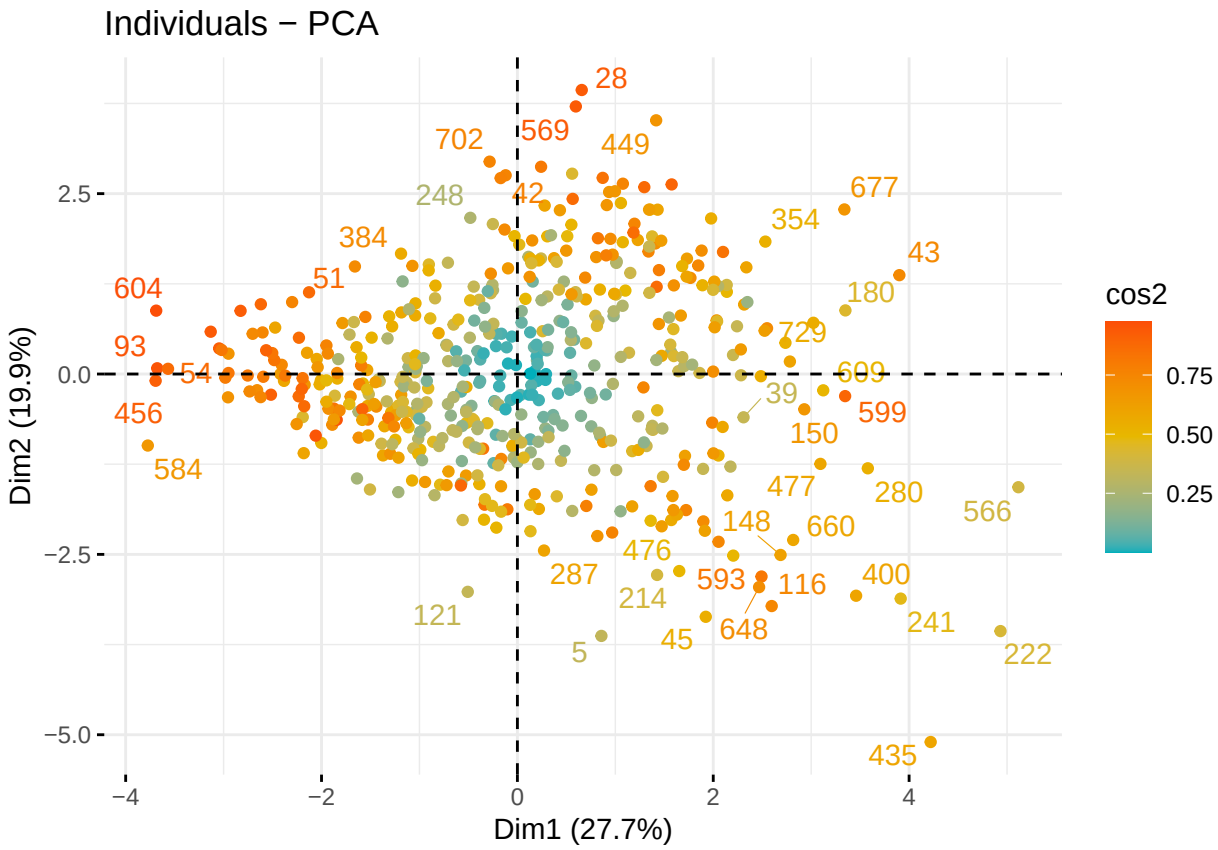


Come si può notare dal grafico, praticamente tutte le variabili sono positivamente correlate dal momento che sono raggruppate in un unico punto. Tuttavia,la variabile `DiabetesPedigreeFunction` è l'unica non correlata con le altre. Inoltre, è quella meglio rappresentata dal momento che risulta più distante della altre dall'origine.

Componenti Principali e individui Per vedere gli effetti delle PCA sugli individui del trainset viene effettuata una proiezione su un grafo:

```
ind = get_pca_ind(res.pca)
fviz_pca_ind(res.pca,
             axes = c(1, 2),
             col.ind = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)
```

```
## Warning: ggrepel: 486 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

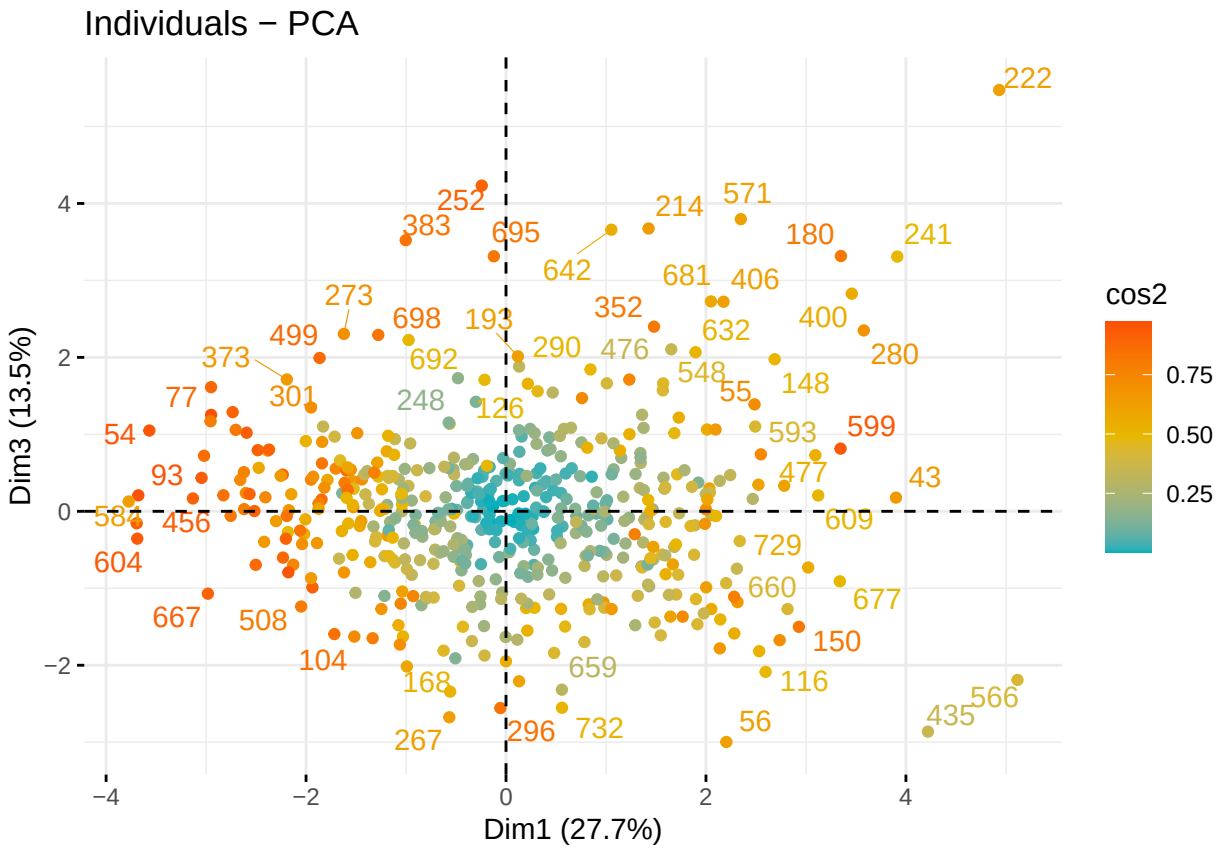


Riguardo le prime due dimensioni, come si può vedere, un alto `cos2`, quindi di colore tra arancione e rosso, indica che gli individui sono ben rappresentati dalle dimensioni 1 e 2. Segue che la maggioranza degli individui sono correttamente rappresentati da queste due componenti.

Viene effettuato il procedimento analogo con la dimensione 1 e 3.

```
ind = get_pca_ind(res.pca)
fviz_pca_ind(res.pca,
             axes = c(1, 3),
             col.ind = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)
```

```
## Warning: ggrepel: 470 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

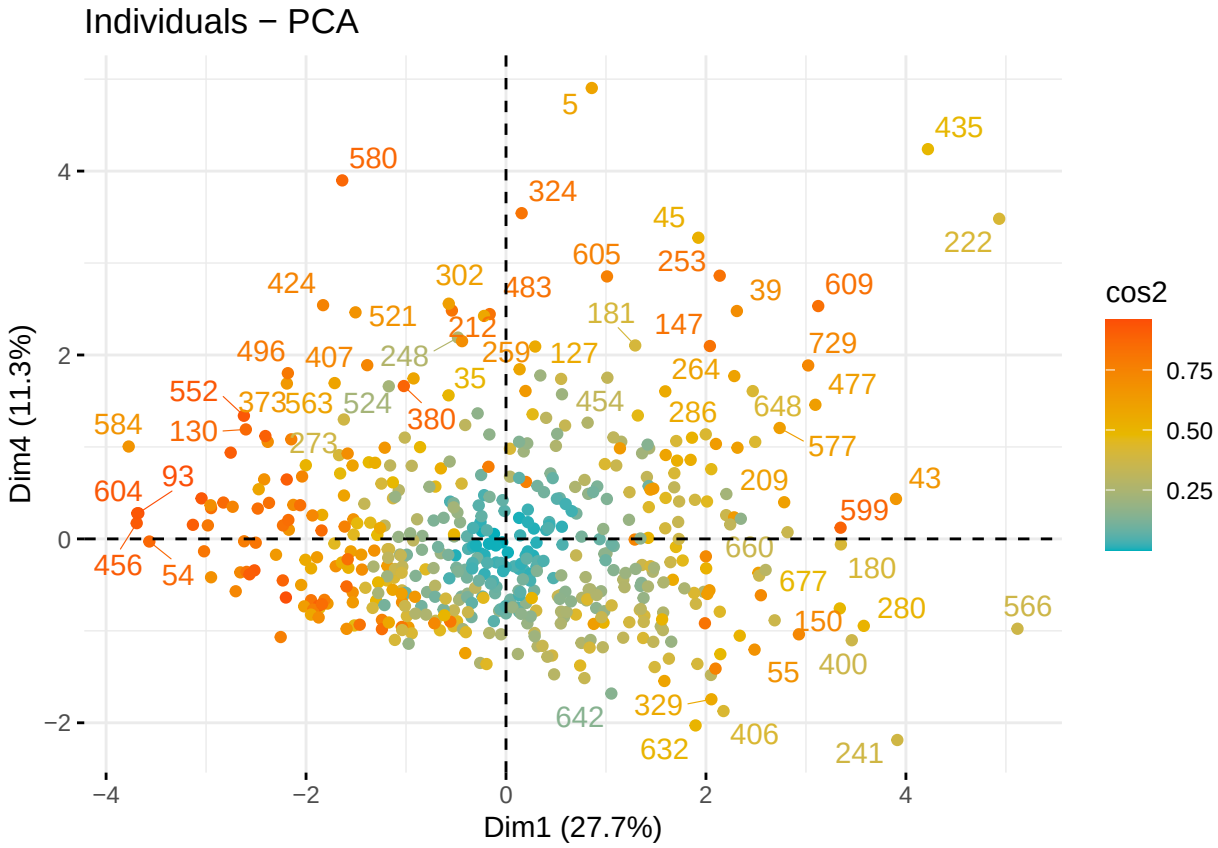


Anche in questo caso il risultato è quasi simile al precedente, anche se peggiore del primo.

Infine, segue il grafico per le dimiensioni 1 e 4:

```
ind = get_pca_ind(res.pca)
fviz_pca_ind(res.pca,
             axes = c(1, 4),
             col.ind = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)
```

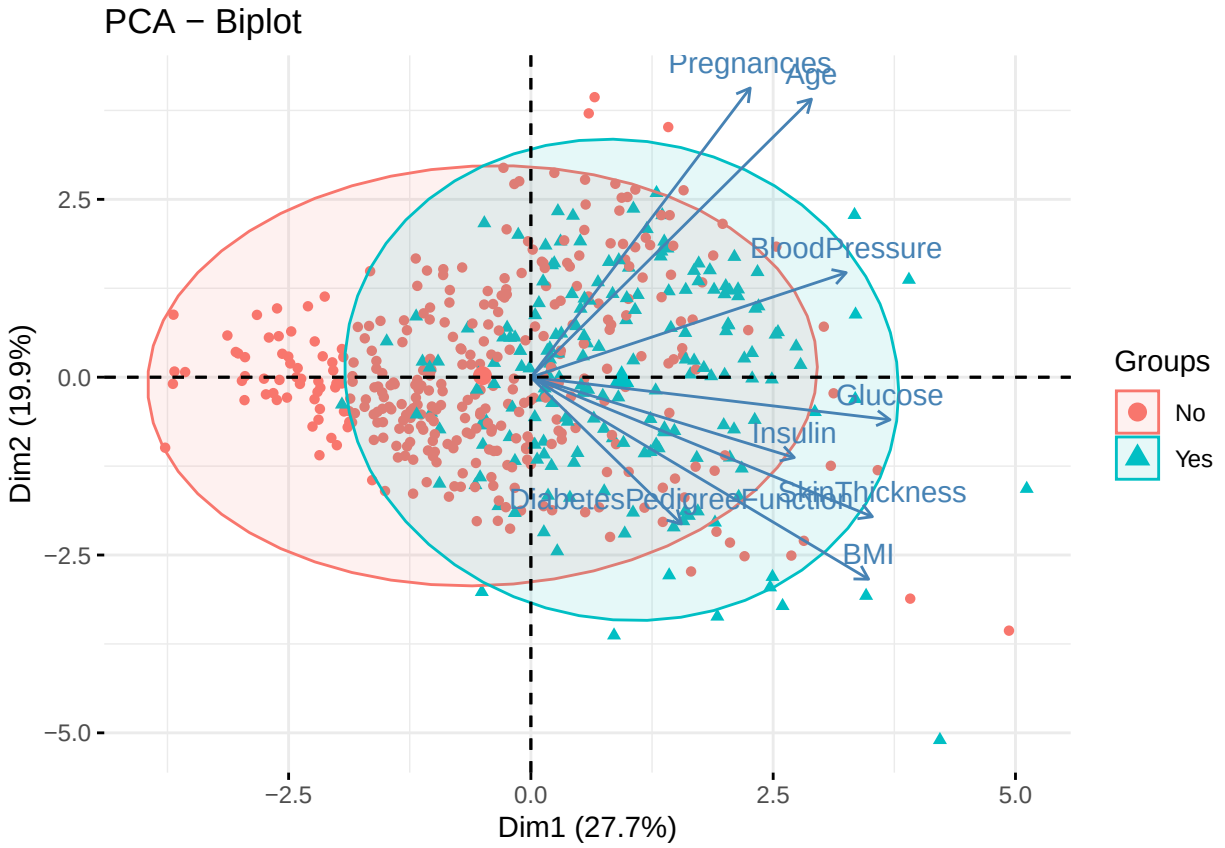
```
## Warning: ggrepel: 468 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



In questo caso vi è una leggera prevalenza del valore `cos2` mediamente alto rispetto al grafico della componente 1 e 3.

In seguito viene riportato il biplot delle prime due componenti principali, raggruppando anche i punti in base alla classe di appartenenza.

```
fviz_pca_biplot(res.pca,
                geom.ind = "point",
                col.ind = trainset$Outcome,
                addEllipses = TRUE,
                legend.title = "Groups")
```



Da questo si nota una sovrapposizione delle classi tramite la rappresentazione delle componenti principali 1 e 2.

Scelta dei modelli ed esperimenti

I due modelli presi in considerazione sono gli alberi di decisione e il Naive Bayes.

Per quanto riguarda la scelta del primo, questa è stata influenzata dalle seguenti motivazioni:

- Risulta molto intuitivo e facile da analizzare e interpretare, ciò permette di illustrare chiaramente i risultati ottenuti da esso. Quindi risulta vantaggioso anche in ambito medico, come nel caso in questione.
- Sono in grado di funzionare discretamente bene anche se le assunzioni fatte sui dati dovessero essere violate.
- Gli alberi decisionali eseguono implicitamente lo screening delle variabili o la selezione delle caratteristiche.
- Gli alberi decisionali richiedono uno sforzo relativamente ridotto da parte degli utenti per la preparazione dei dati.

Inoltre, è stato preso in considerazione il fatto che l'albero decisionale ha il rischio di essere troppo specifico per il trainset (overfitting). Questo è stato gestito durante lo sviluppo di tale modello.

Per quanto riguarda l'utilizzo del modello naive Bayes viene motivato dai seguenti punti:

- Uno dei principali vantaggi del Classificatore Naive Bayes è il funzionamento ottimale anche con un set di addestramento di dimensioni ridotte. Questo vantaggio deriva dal fatto che il classificatore Naive Bayes è parametrizzato tramite la media e la varianza di ogni variabile indipendentemente da tutte le altre variabili (6).

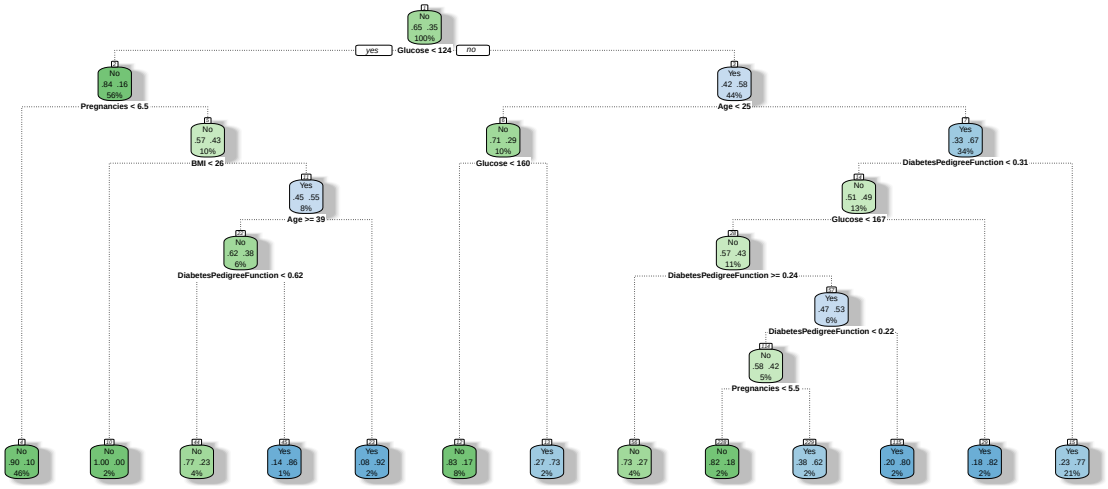
- Algoritmo molto semplice e veloce. Funziona bene sia sulla classificazione binaria sia su quella multi-classe (7).
- Esso esplicita la manipolazione delle probabilità ed è tra gli approcci più pratici a certi tipi di problemi di apprendimento (infatti Bayes compete con gli alberi di decisione e le reti neurali).
- È specificato che le variabili sono tra loro indipendenti e che i valori assunti dalla variabile target sono presi da un insieme di valori finiti.

Decision Tree

In questo paragrafo verrà mostrato l'allenamento dell'albero di decisione e una eventuale potatura dello stesso. Si noti che i seguenti modelli vengono allenati tutti con una 10-fold cross validation.

L'albero viene allenato sul training set tramite i seguenti comandi:

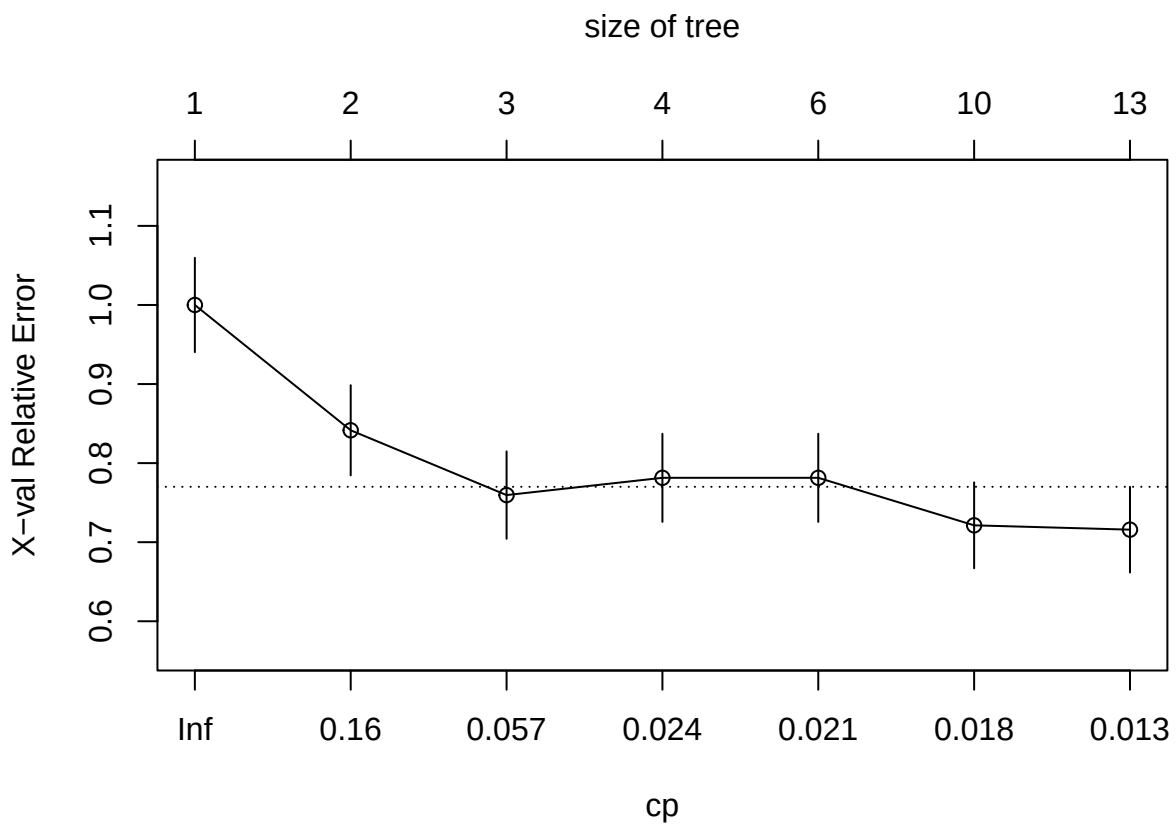
```
set.seed(1)
dt = rpart(Outcome ~ .,
           data = trainset,
           method = "class",
           control = rpart.control(xval=10))
fancyRpartPlot(dt)
```



Rattle 2023−Gen−22 17:08:48 antoniogrillo

Come si può notare, l'albero risultante è folto e potrebbe necessitare di una potatura per evitare l'overfitting dei dati. Per poterlo potare, si dovrà stimare il corretto parametro di complessità `cp`. Infatti questo parametro permette la terminazione della generazione di nuovi rami dell'albero una volta superato. Per stimarlo, viene mostrato il seguente grafico dei vari `cp` che possono essere assunti dall'albero.

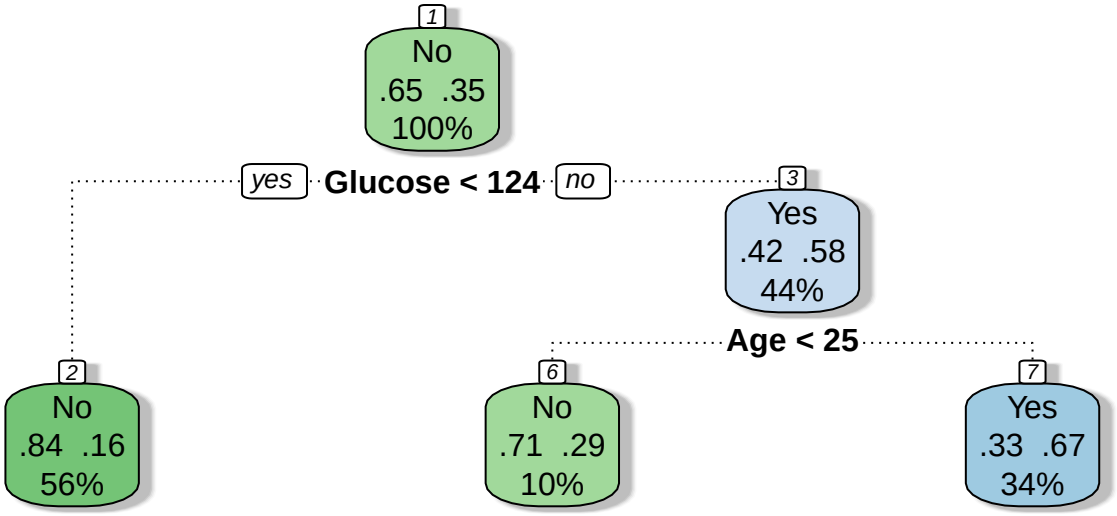
```
plotcp(dt)
```



Secondo la documentazione, il `cp` ottimale da scegliere è `cp = 0.057`.

Quindi effettuo una potatura sull'albero impostando l'opportuno `cp`.

```
pdt = rpart(Outcome ~ .,
            data = trainset,
            method = "class",
            control = rpart.control(xval=10,
                                    cp = 0.057))
fancyRpartPlot(pdt)
```



Una volta ottenuti i due alberi, vanno confrontati e misurate le performance di entrambi per poter scegliere il migliore.

Confronti tra alberi Matrice di confusione per l'albero non potato:

```
dtp = predict(dt, testset, type = "class")
rdt = confusionMatrix(dtp,
                      testset$Outcome,
                      mode = "prec_recall")
rdt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  120  34
##        Yes  24  47
##
##                Accuracy : 0.7422
##                  95% CI : (0.6799, 0.7981)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.0006944
##
##                   Kappa : 0.4251
##
##  Mcnemar's Test P-Value : 0.2373018
##
##               Precision : 0.7792
##                  Recall : 0.8333
##                      F1 : 0.8054
##              Prevalence : 0.6400
##          Detection Rate : 0.5333
##    Detection Prevalence : 0.6844
```

```
##       Balanced Accuracy : 0.7068
##
##        'Positive' Class : No
##
```

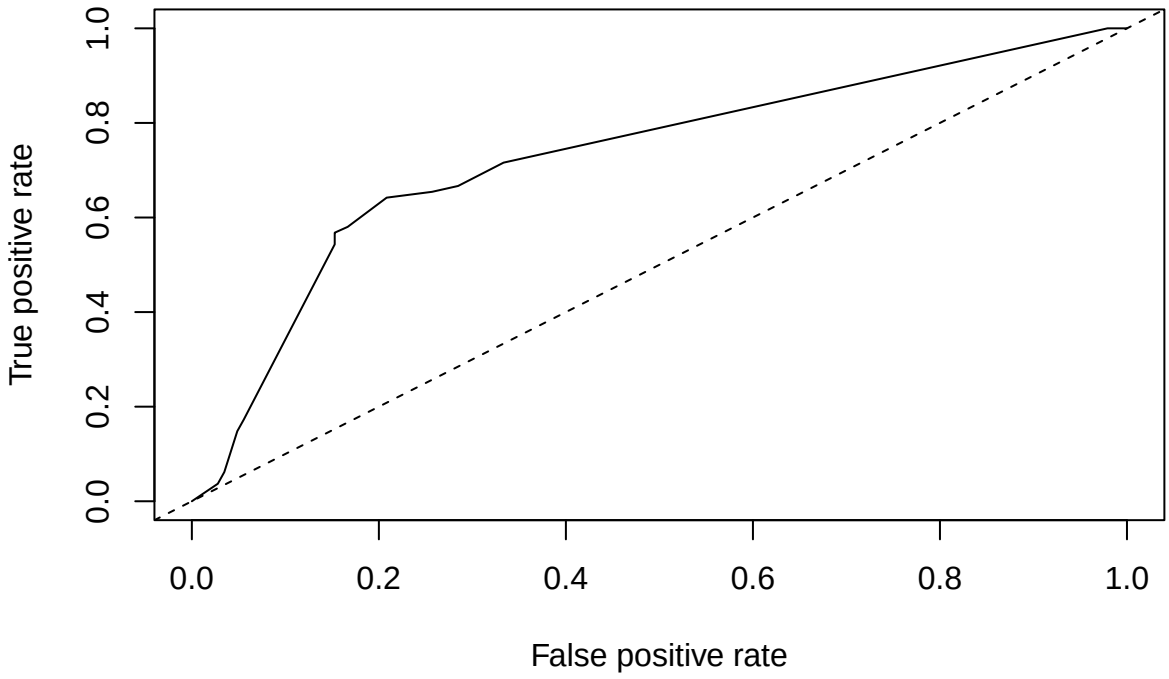
Segue la matrice di confusione dell'albero potato:

```
pdtp = predict(pdt, testset, type = "class")
rpdt = confusionMatrix(pdtp,
                       testset$Outcome,
                       mode = "prec_recall")
rpdt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  115  35
##        Yes  29  46
##
##                Accuracy : 0.7156
##                  95% CI : (0.6518, 0.7735)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.01005
##
##                   Kappa : 0.3725
##
##  Mcnemar's Test P-Value : 0.53197
##
##               Precision : 0.7667
##                  Recall : 0.7986
##                      F1 : 0.7823
##              Prevalence : 0.6400
##          Detection Rate : 0.5111
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 0.6833
##
##        'Positive' Class : No
##
```

Viene mostrata ora la curva ROC dell'albero di decisone originale:

```
dt.pred.cart = predict(dt,
                       newdata = testset,
                       type = "prob")[, 2]
dt.pred.rocr = prediction(dt.pred.cart, testset$Outcome)
dt.perf.tpr.rocr = performance(dt.pred.rocr, "tpr", "fpr")
dt.perf.rocr = performance(dt.pred.rocr,
                        measure = "auc",
                        x.measure = "cutoff")
plot(dt.perf.tpr.rocr,
     main=paste("DT AUC:",(dt.perf.rocr@y.values)))
abline(0, 1, lty = 2)
```



Di seguito viene definita la funzione per determinare il cut-off ottimale:

```
opt.cut = function(perf, pred){
  cut.ind = mapply(FUN=function(x, y, p){
    d = (x - 0)^2 + (y-1)^2
    ind = which(d == min(d))
    c(sensitivity = y[[ind]],
      specificity = 1-x[[ind]],
      cutoff = p[[ind]])
  }, perf@x.values, perf@y.values, pred@cutoffs)
}
```

Viene stampato il Cut-Off ottimale:

```
print("DT")
```

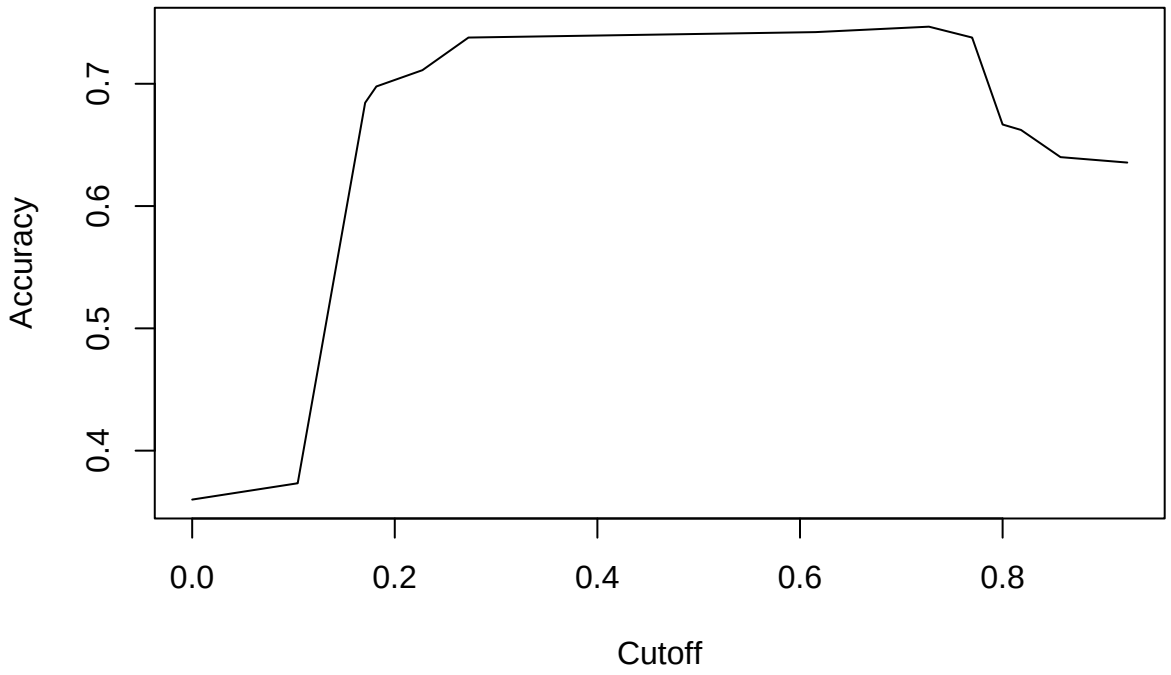
```
## [1] "DT"
```

```
print(opt.cut(dt.perf.tpr.rocr, dt.pred.rocr))
```

```
##                 [,1]
## sensitivity 0.6419753
## specificity 0.7916667
## cutoff      0.2727273
```

Quindi si riporta il grafico di confronto tra cut-off e accuracy:

```
dt.acc.perf = performance(dt.pred.rocr, measure = "acc")
plot(dt.acc.perf)
```



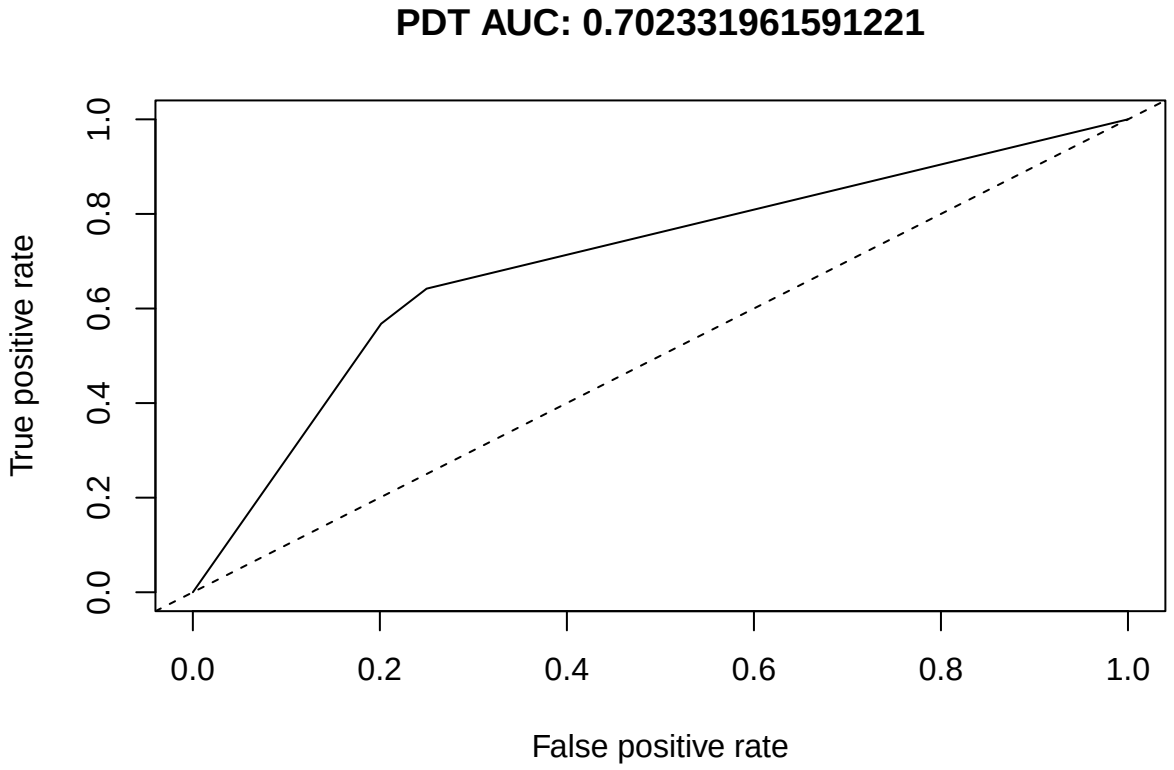
Infine, l'overall accuracy risulta essere:

```
ind = which.max(slot(dt.acc.perf, "y.values")[[1]] )
dt.acc = slot(dt.acc.perf, "y.values")[[1]][ind]
cutoff = slot(dt.acc.perf, "x.values")[[1]][ind]
print(c(accuracy = dt.acc, cutoff = cutoff))
```

```
##   accuracy cutoff.231
##  0.7466667  0.7272727
```

Di seguito vengono effettuate le procedure analoghe anche per l'albero potato. Si inizia con la curva ROC.

```
pdt.pred.cart = predict(pdt,
                        newdata = testset,
                        type = "prob")[, 2]
pdt.pred.rocr = prediction(pdt.pred.cart, testset$Outcome)
pdt.perf.tpr.rocr = performance(pdt.pred.rocr, "tpr", "fpr")
pdt.perf.rocr = performance(pdt.pred.rocr,
                        measure = "auc",
                        x.measure = "cutoff")
plot(pdt.perf.tpr.rocr,
     main=paste("PDT AUC:",(pdt.perf.rocr@y.values)))
abline(0, 1, lty = 2)
```



Segue il cut-off ottimale:

```
print("PDT")
```

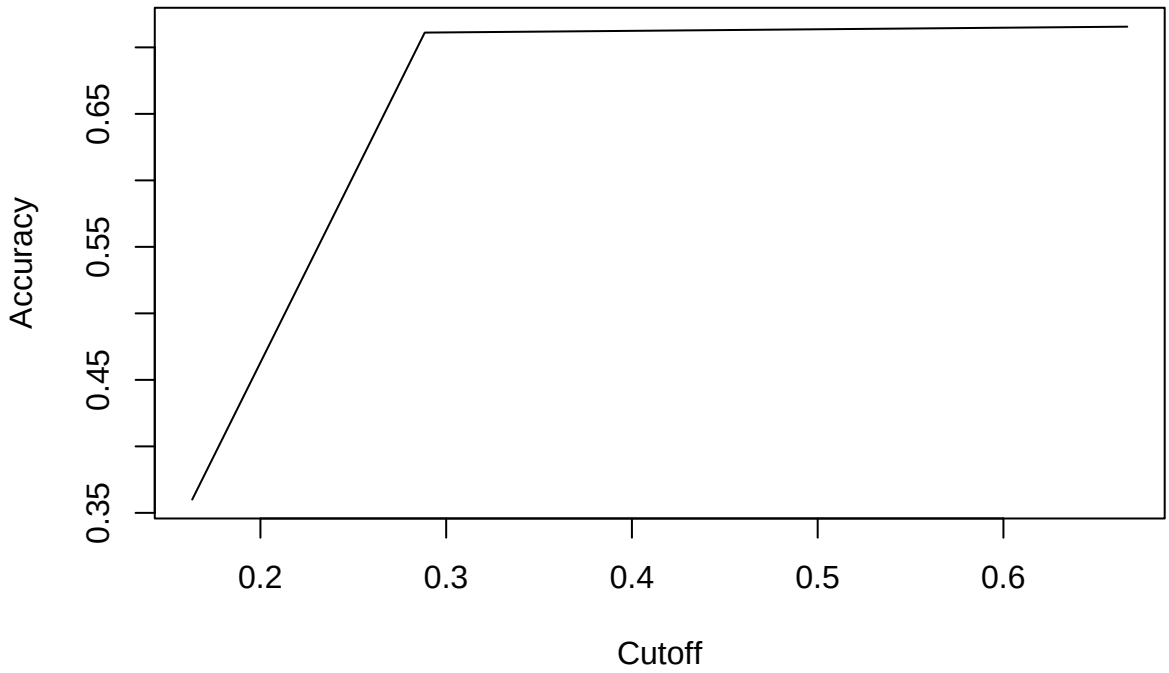
```
## [1] "PDT"
```

```
print(opt.cut(pdt.perf.tpr.rocr, pdt.pred.rocr))
```

```
##                     [,1]
## sensitivity 0.6419753
## specificity 0.7500000
## cutoff      0.2884615
```

E il confronto accuracy-cut-off:

```
pdt.acc.perf = performance(pdt.pred.rocr, measure = "acc")
plot(pdt.acc.perf)
```



Infine, viene stampato il dato sulla overall accuracy:

```
ind = which.max(slot(pdt.acc.perf, "y.values")[[1]])
pdt.acc = slot(pdt.acc.perf, "y.values")[[1]][ind]
cutoff = slot(pdt.acc.perf, "x.values")[[1]][ind]
print(c(accuracy = pdt.acc, cutoff = cutoff))
```

```
##  accuracy cutoff.75
## 0.7155556 0.6666667
```

Analisi e conclusioni In questa sezione verranno confrontati i risultati delle performance dei due alberi e, in base ad essi, verrà scelto il migliore.

Principalmente, vengono confrontate le matrici di confusione dei due alberi:

```
print("Albero normale: ")
```

```
## [1] "Albero normale: "
```

```
rdt$table
```

```
##          Reference
## Prediction  No Yes
##        No  120  34
##        Yes  24  47
```

```
print('------------------')
```

```
## [1] "------------------"
```

```
print("Albero pruned: ")
```

```
## [1] "Albero pruned: "
```

```
rpdt$table
```

```
##          Reference
```

```
## Prediction  No Yes
##        No  115  35
##        Yes  29  46
```

Analizzando le matrici di confusione si nota che l'albero pruned ha qualche errore in più nella predizione rispetto al non pruned. Infatti, si noti che il numero dei falsi positivi del pruned è maggiore di quello dell'albero normale.

L'accuracy dei due modelli è la seguente:

```
print("Albero normale: ")
```

```
## [1] "Albero normale: "
```

```
rdt$overall['Accuracy']
```

```
##  Accuracy
## 0.7422222
```

```
print('-----------------')
```

```
## [1] "-----------------"
```

```
print("Albero pruned: ")
```

```
## [1] "Albero pruned: "
```

```
rpdt$overall['Accuracy']
```

```
##  Accuracy
## 0.7155556
```

Da ciò segue che l'albero non potato ha una accuracy leggermente maggiore rispetto al pruned. Tuttavia, essendo il test set sbilanciato in quanto ci sono molti più individui con `Outcome = No` rispetto a quelli con `Yes`, l'accuracy in questo caso potrebbe non essere una metrica significativa.

Vengono mostrate le misure di precision e recall:

```
print("Albero normale: ")
```

```
## [1] "Albero normale: "
```

```
rdt$byClass['Precision']
```

```
## Precision
## 0.7792208
```

```
print('-----------------')
```

```
## [1] "-----------------"
```

```
print("Albero pruned: ")
```

```
## [1] "Albero pruned: "
```

```
rpdt$byClass['Precision']
```

```
## Precision
## 0.7666667
```

Le precision risultanti hanno valori quasi simili. Quindi si può affermare che entrambi gli alberi hanno una precisione simile nel predirre le classi positive del test set. Questo porta a preferire l'albero non potato per il fatto che la sua precision è leggermente più alta dell'altro.

```
print("Albero normale: ")
```

```
## [1] "Albero normale: "
```

```
rdt$byClass['Recall']
```

```
##    Recall
## 0.8333333
```

```
print('------------------')
```

```
## [1] "------------------"
```

```
print("Albero pruned: ")
```

```
## [1] "Albero pruned: "
```

```
rpdt$byClass['Recall']
```

```
##    Recall
## 0.7986111
```

Da quanto riportato, si può esservare che l'albero non potato ha una recall più alta di quello potato. Questo è un parametro di performance da tenere in considerazione soprattutto in ambito medico, visto che è importante essere sicuri che ogni minimo esempio considerato positivo da parte del modello venga sottoposto a ispezione umana (come nel caso in questione). Da ciò si può dedurre che è preferibile l'albero non potato.

Segue anche la F-measure:

```
print("Albero normale: ")
```

```
## [1] "Albero normale: "
```

```
rdt$byClass['F1']
```

```
##        F1
## 0.8053691
```

```
print('------------------')
```

```
## [1] "------------------"
```

```
print("Albero pruned: ")
```

```
## [1] "Albero pruned: "
```

```
rpdt$byClass['F1']
```

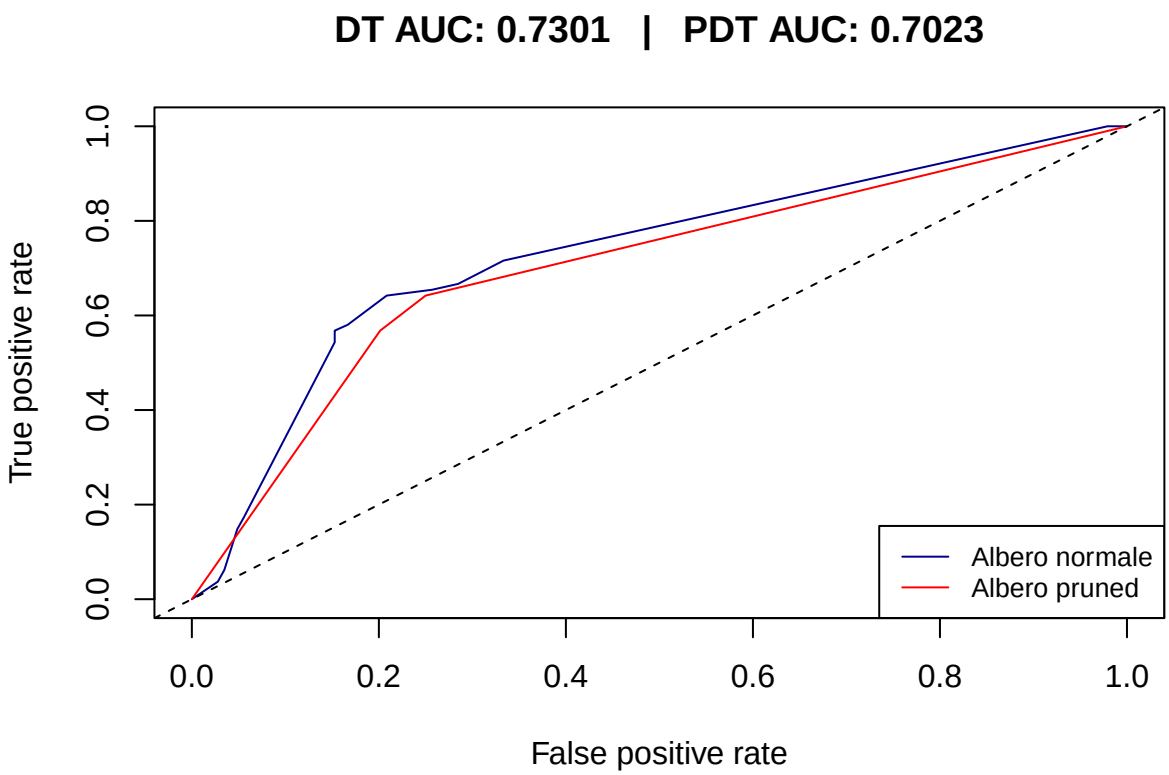
```
##        F1
## 0.7823129
```

L'albero non pruned ha un F1 maggiore rispetto all'altro. Questa misura riassume le misure di precision e recall essendo una media armonica delle due, e, anche se di poco, si può affermare che il primo albero sia preferibile al secondo.

Seguono le curve ROC degli alberi:

```
plot(dt.perf.tpr.rocr,
     col = "darkblue",
     main=paste("DT AUC:",
                round((dt.perf.rocr@y.values[[1]]), 4),
                "  |  ",
                "PDT AUC:",
```

```
                round(pdt.perf.rocr@y.values[[1]], 4)))
plot(pdt.perf.tpr.rocr,
     add = TRUE,
     col = "red")
abline(0, 1, lty = 2)
legend("bottomright", legend=c("Albero normale", "Albero pruned"),
       col=c("darkblue", "red"), lty=1, cex=0.8)
```



Da questo grafico si nota che l'albero normale ha una AUC maggiore rispetto al pruned. Ciò significa che si avvicina di più a quello che dovrebbe essere un classificatore ottimale.

In conclusione, vista l'importanza in questo campo della misura di recall e viste le curve ROC con relativa area sotto la curva (AUC), si è deciso di utilizzare l'albero normale rispetto al pruned come modello.

Naive Bayes

Anche per bayes viene impostata una 10-fold cross validation:

```
control = trainControl(method = "cv",
                       number = 10,
                       classProbs = TRUE,
                       summaryFunction = twoClassSummary)
```

Alleno il modello bayesiano:

```
bayes.model = train(Outcome ~ .,
                    data = trainset,
                    method = "naive_bayes",
                    metric = "ROC",
                    trControl = control)
```

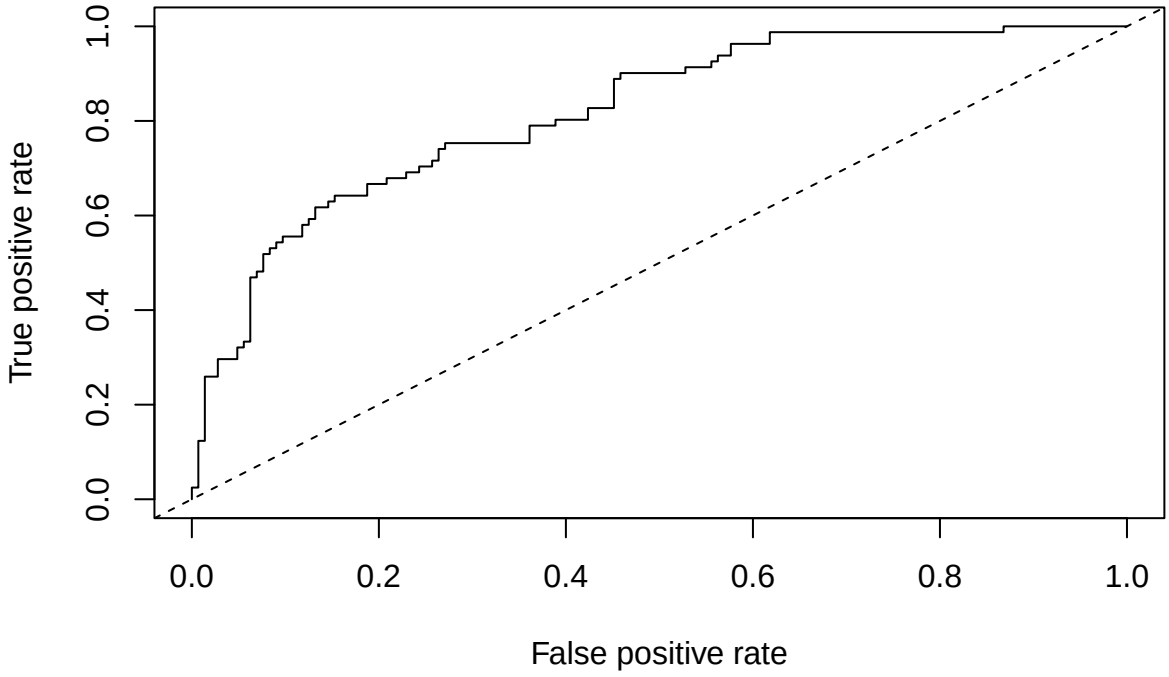
Mostro la matrice di confusione ottenuta con altre misure di performance:

```
nbp = predict(bayes.model, testset)
resultBayes = confusionMatrix(nbp,
                              testset$Outcome,
                              mode = "prec_recall")
resultBayes
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  125  31
##        Yes  19  50
##
##                Accuracy : 0.7778
##                  95% CI : (0.7177, 0.8303)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 5.726e-06
##
##                   Kappa : 0.5016
##
##  Mcnemar's Test P-Value : 0.1198
##
##               Precision : 0.8013
##                  Recall : 0.8681
##                      F1 : 0.8333
##              Prevalence : 0.6400
##          Detection Rate : 0.5556
##    Detection Prevalence : 0.6933
##       Balanced Accuracy : 0.7427
##
##        'Positive' Class : No
##
```

Ora viene mostrata la relativa curva ROC con la AUC:

```
nb = bayes.model
nb.pred.nb = predict(nb, newdata = testset, type = "prob")[,2]
nb.pred.rocr = prediction(nb.pred.nb, testset$Outcome)
nb.perf.tpr.rocr = performance(nb.pred.rocr, "tpr", "fpr")
nb.perf.rocr = performance(nb.pred.rocr, measure = "auc", x.measure = "cutoff")
plot(nb.perf.tpr.rocr, main=paste("AUC:",(nb.perf.rocr@y.values)))
abline(0, 1, lty = 2)
```



Segue il cut-off ottimale:

```
print("Bayes cut-off")
```

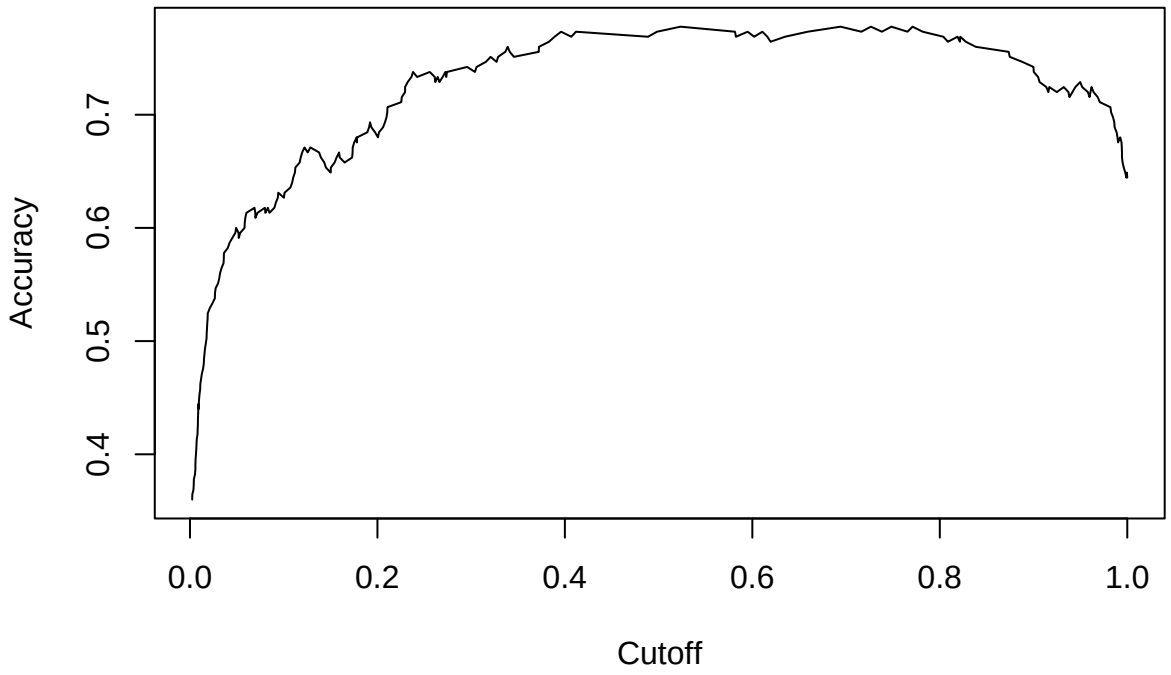
```
## [1] "Bayes cut-off"
```

```
print(opt.cut(nb.perf.tpr.rocr, nb.pred.rocr))
```

```
##                     [,1]
## sensitivity 0.7530864
## specificity 0.7291667
## cutoff      0.2380011
```

E il confronto accuracy-cut-off:

```
nb.acc.perf = performance(nb.pred.rocr, measure = "acc")
plot(nb.acc.perf)
```



Infine, viene stampato il dato sulla overall accuracy:

```
ind = which.max(slot(nb.acc.perf, "y.values")[[1]])
nb.acc = slot(nb.acc.perf, "y.values")[[1]][ind]
cutoff = slot(nb.acc.perf, "x.values")[[1]][ind]
print(c(accuracy = nb.acc, cutoff = cutoff))
```

```
##  accuracy    cutoff
## 0.7777778 0.7709248
```

Analisi risultati

In questa sezione verranno confrontati i risultati delle performance dei due modelli e, in base ad essi, verrà scelto il migliore.

Principalmente, vengono confrontate le matrici di confusione dell'albero di decisione e del classificatore bayesian:

```
print("Albero: ")
```

```
## [1] "Albero: "
```

```
rdt$table
```

```
##          Reference
## Prediction  No Yes
##        No  120  34
##        Yes  24  47
```

```
print('-----------------')
```

```
## [1] "-----------------"
```

```
print("Naive Bayes: ")
```

```
## [1] "Naive Bayes: "
```

```
resultBayes$table
```

```
##            Reference
## Prediction  No Yes
##        No  125  31
##        Yes  19  50
```

Analizzando le matrici di confusione, si nota che l'albero ha qualche errore in più nella predizione rispetto al naive Bayes. Infatti, si noti che il numero dei falsi positivi del bayesian è minore di quello dell'albero.

L'accuracy dei due modelli è la seguente:

```
print("Albero: ")
```

```
## [1] "Albero: "
```

```
rdt$overall['Accuracy']
```

```
##  Accuracy
## 0.7422222
```

```
print('------------------')
```

```
## [1] "------------------"
```

```
print("Naive Bayes: ")
```

```
## [1] "Naive Bayes: "
```

```
resultBayes$overall['Accuracy']
```

```
##  Accuracy
## 0.7777778
```

Da ciò segue che l'albero ha una accuracy minore rispetto al bayesian. Tuttavia, essendo il test set sbilanciato, in quanto ci sono molti più individui con `Outcome = No` rispetto a quelli con `Yes`, l'accuracy in questo caso potrebbe non essere una metrica significativa.

Vengono mostrate le misure di precision e recall:

```
print("Albero: ")
```

```
## [1] "Albero: "
```

```
rdt$byClass['Precision']
```

```
## Precision
## 0.7792208
```

```
print('------------------')
```

```
## [1] "------------------"
```

```
print("Naive Bayes: ")
```

```
## [1] "Naive Bayes: "
```

```
resultBayes$byClass['Precision']
```

```
## Precision
## 0.8012821
```

Le precision risultanti hanno poco discostanti. Quindi si può affermare che entrambi i modelli hanno una precisione quasi simile nel predirre le classi positive del test set. Questo porta a preferire il Naive Bayes per il fatto che la sua precision è leggermente più alta dell'altro.

```
print("Albero: ")
```

```
## [1] "Albero: "
```

```
rdt$byClass['Recall']
```

```
##    Recall
## 0.8333333
```

```
print('-----------------')
```

```
## [1] "-----------------"
```

```
print("Naive Bayes: ")
```

```
## [1] "Naive Bayes: "
```

```
resultBayes$byClass['Recall']
```

```
##    Recall
## 0.8680556
```

Da quanto riportato, si può esservare che il classificatore Naive Bayes ha una recall più alta dell'albero di decisione. Come già espresso in precedenza, questo è un parametro di performance da tenere in considerazione soprattutto in ambito medico, visto che è importante essere sicuri che ogni minimo esempio considerato positivo da parte del modello venga sottoposto a ispezione umana (come nel caso in questione). Da ciò si può dedurre che è preferibile il Naive Bayes.

Segue anche la F-measure:

```
print("Albero: ")
```

```
## [1] "Albero: "
```

```
rdt$byClass['F1']
```

```
##        F1
## 0.8053691
```

```
print('-----------------')
```

```
## [1] "-----------------"
```

```
print("Naive Bayes: ")
```

```
## [1] "Naive Bayes: "
```

```
resultBayes$byClass['F1']
```

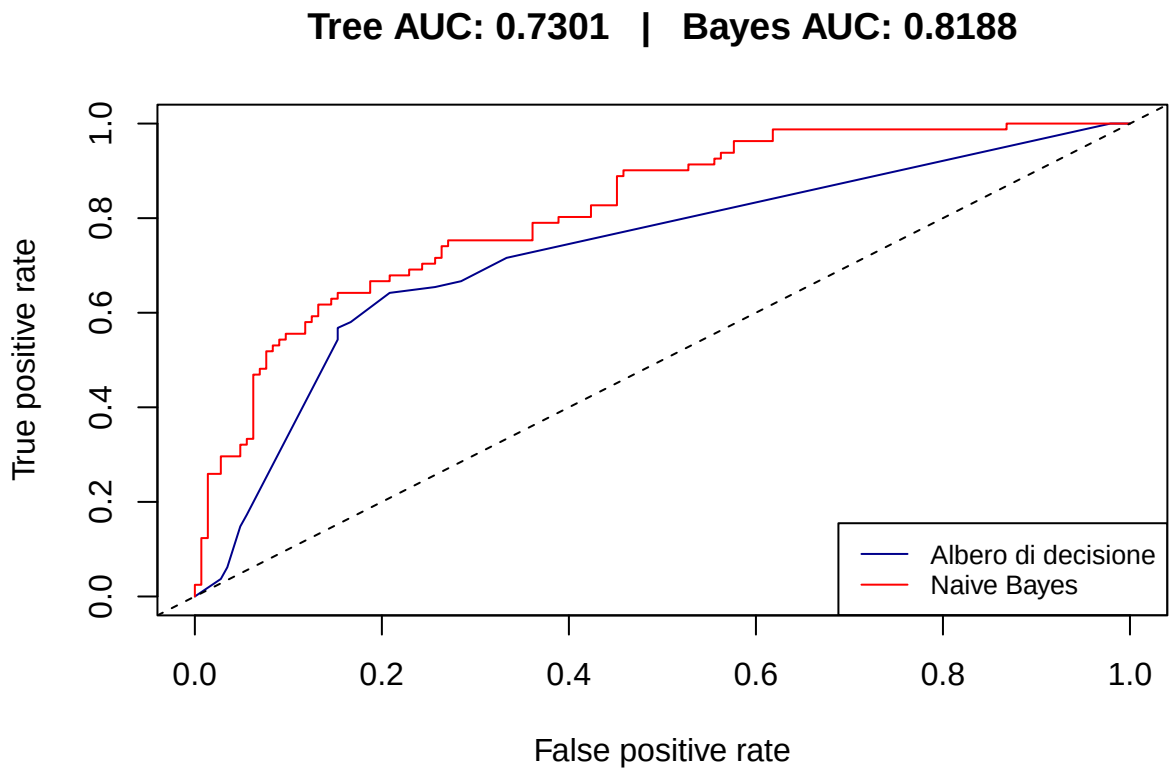
```
##        F1
## 0.8333333
```

L'albero ha un F1 minore rispetto a Naive Bayes. Questa misura riassume le misure di precision e recall essendo una media armonica delle due, e, anche se di poco, si può affermare che il primo albero sia preferibile al secondo.

Seguono le curve ROC dei modelli:

```
plot(dt.perf.tpr.rocr,
     col = "darkblue",
```

```
     main=paste("Tree AUC:",
                round((dt.perf.rocr@y.values[[1]]), 4),
                "  |  ",
                "Bayes AUC:",
                round(nb.perf.rocr@y.values[[1]], 4)))
plot(nb.perf.tpr.rocr,
     add = TRUE,
     col = "red")
abline(0, 1, lty = 2)
legend("bottomright", legend=c("Albero di decisione",
                               "Naive Bayes"),
       col=c("darkblue", "red"),
       lty=1, cex=0.8)
```



Da questo grafico si nota che l'albero normale ha una AUC minore rispetto al bayes. Ciò significa che si avvicina di più a quello che dovrebbe essere un classificatore ottimale.

Conclusioni

Nell'analisi di questo dataset, si è deciso di utilizzare e confrontare due modelli di machine learning: alberi di decisione e naive Bayes.

Si sono svolti degli esperimenti su questi modelli, in particolare usando una 10-fold cross validation e illustrando le varie misure di performance.

Inoltre, sono stati svolti degli esperimenti aggiuntivi anche sull'albero di decisione. Infatti, questi sono serviti per evitare l'overfitting dell'albero. Applicando i criteri di potatura e confrontando l'albero potato e quello normale, è risultato che l'albero normale è il migliore e quindi non c'è un rischio di sovradattare il dataset.

In seguito, si sono confrontati i risultati degli esperimenti fatti con l'albero migliore e il classificatore Naive Bayes.

L'analisi delle misure di performance riporta che il miglior classificatore tra i due è il naive Bayes. Infatti, esso possiede la maggiore precision, recall (molto importante essendo in ambito medico) e AUC della relativa curva ROC.